MASTER EN SCIENCES INFORMATIQUES

La programmation par contraintes: une solution au problème de la constitution des horaires de sessions d'examens à l'université ?

Vandeput, Etienne

Award date:
2003

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 2002-2003

La programmation par contraintes: une solution au problème de la constitution des horaires de sessions d'examens à l'université?

Etienne Vandeput

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique

Résumé

Les problèmes de planification, et singulièrement, les problèmes de conception d'horaires ont vite été perçus par les informaticiens comme des problèmes potentiellement solubles par des programmes informatiques. On sait que leur complexité est telle, que les techniques basées sur la génération et la vérification des solutions conduisent à des temps de réponse des programmes qui sont infinis à l'échelle humaine. La technique de coloration des graphes et ses algorithmes sont bien connus de ceux qui tentent de les résoudre. Le travail relaté dans cet ouvrage s'intéresse à une autre voie, celle de la programmation par contraintes, dans le but de mener à bien la tâche de constitution d'un horaire de session d'examen à l'université. Le contexte décrit est suffisamment général pour s'appliquer à d'autres institutions que celle pour laquelle le programme d'accompagnement a été mis au point. La réflexion ne se limite pas à la partie traitement mais porte également sur la constitution d'une interface ergonomique de gestion des données.

Mots-clés: *horaire, planification, programmation par contraintes, solveur, PLC, HCLP, coloration des graphes*

Abstract

Scheduling problems, particularly timetables conception difficulties, quickly appeared to computer specialists' eyes like problems that could rather easily be solved by data processing programs. Their complexity is such that trying to produce solutions with "generate and test" methods may cause unlimited response times. Graph coloring technique and its algorithms are well known in that domain, but the contents of this book show another way, constraint programming, with purpose to build an examination schedule at university. The described context is quite general, so that the helpful joined program can easily be used by similar organizations. This work does not only concern processing but goes further about the building of an ergonomic interface for data management.

Keywords: *scheduling, constraint programming, solver, CLP, HCLP, labeling, graph coloring*

Table des matières

Glossaire

Arc: dans la théorie des graphes, lien orienté entre deux sommets ou noeuds

Arête: dans la théorie des graphes, lien non orienté entre deux sommets ou noeuds

Backtracking: (fr: parcours par retour arrière) en programmation logique notamment, processus élémentaire d'exploration des arbres en vue de la recherche d'une solution

CLP: (Constraint Logic Programming ou programmation logique par *contraintes*) schème associant le caractère déclaratif de la programmation logique aux fonctionnalités d'un *solveur* de contraintes et regroupant divers langages en fonction du domaine des variables auxquelles ces contraintes s'appliquent

Coloration des (sommets des) graphes: technique consistant à attribuer une couleur à chaque sommet d'un graphe en évitant que deux *sommets* adjacents soient de couleur identique

Comparateur: paramètre, dans le schème *HCLP*, qui détermine comment deux *valuations* peuvent être comparées; il existe des comparateurs locaux, globaux et régionaux selon la manière dont les *contraintes* d'un même niveau de la hiérarchie sont considérées

Consistance: la consistance aux *noeuds* assure que les valeurs possibles des *sommets* d'un graphe respectent les *contraintes* unaires qui les concernent; la consistance aux arcs assure que les valeurs possibles des sommets pris deux à deux respectent les contraintes binaires qui les concernent

Contrainte: une contrainte peut être vue comme une relation entre deux variables qui en limite les valeurs possibles; une contrainte douce ne doit pas nécessairement être prise en compte par le *solveur* de contraintes; une contrainte indirecte est indépendante du contexte examiné et traduit souvent une évidence

Fonction d'agrégation: une fonction qui détermine comment le respect des *contraintes* d'un même niveau est évalué globalement (somme des erreurs, erreur maximum, somme des carrés des erreurs,...)

Forward checking: technique de propagation des *contraintes* dont le principe est de vérifier, lors de l'attribution d'une valeur à une variable, si toutes les variables qui lui sont liées peuvent être affectées

Generate-and-test (GT): méthode naïve consistant à donner une valeur à chaque variable avant de tester si la solution convient

GNU-Prolog: compilateur *Prolog* libre de droits et gratuit conforme à l'*ISO* standard pour Prolog

HCLP: (Hierarchical Constraint Logic Programming ou programmation logique par hiérarchie de contraintes) schème incluant la programmation logique et la résolution des *contraintes*, y compris les contraintes douces; comme *CLP*, il est paramétré par un domaine mais également par un *comparateur* pour la résolution de ces contraintes douces

Heuristique: technique essentiellement basée sur l'expérience et les résultats acquis et qui, de ce fait, n'offre pas, les mêmes garanties que l'algorithme

ISO: organisme international de standardisation

Labélisation (labeling): processus de fixation des valeurs aux variables (parmi les valeurs qui sont acceptables)

Look ahead: technique de propagation des *contraintes* dont le principe est de vérifier, lors de l'attribution d'une valeur à une variable, si toutes les variables qui lui sont liées et celles qui sont liées à ces dernières peuvent être affectées (maintien de la *consistance* aux arcs)

MySQL: SGBD développé spécialement pour l'Internet et dont les affinités avec *PHP* permettent la programmation de pages web dynamiques

Noeud: dans un graphe, point de départ et/ou d'aboutissement d'un *arc* ou d'une *arête*

np-complet: qualificatif de complexité d'un problème

PHP: langage de script côté serveur pour la gestion de pages web dynamiques

Prolog: langage générique de programmation logique

Séquence d'erreurs: dans la théorie *HCLP*, suite de valeurs correspondants aux erreurs liées au non-respect des *contraintes* d'un même niveau

Solveur: algorithme testant si une ou des *contraintes* peuvent être satisfaites

Sommet: voir *noeud*

Unaire: se dit d'une *contrainte* qui ne concerne qu'une seule variable

Utilisabilité: notion chère aux concepteurs des interfaces homme-machine et qui s'intéresse à la relative aisance qu'éprouvent les utilisateurs à se servir d'une interface ainsi qu'à leur envie de persévérer dans cette utilisation

Valuation: ensemble de valeurs, une pour chacune des variables apparaissant dans les *contraintes*, choisies dans leurs domaines respectifs

Variable de domaine fini (variable FD): variable dont l'ensemble des valeurs possibles est fini

Le péril s'évanouit quand on ose le regarder.
(Châteaubriand)

Avant-propos

Nous présentons les résultats d'une aventure qui, sans être un péril, ressemble à l'ascension d'une vieille montagne ayant englouti plus d'un alpiniste audacieux. Qu'il nous soit permis d'adresser nos plus vifs remerciements à Monsieur Jean-Marie Jacquet pour les conseils judicieux qu'il nous a prodigués et pour sa disponibilité. Nous aimerions également remercier chaleureusement Madame Gyselle Henrard-Detroye pour les précieuses informations qu'elle nous a transmises. Elle l'a fait tantôt au prix d'un travail supplémentaire de collecte de données, tantôt en consacrant une partie de son temps en discussions qui ont éclairé notre démarche. Quant à nos proches, s'ils n'ont pas participé directement à cette aventure, ils ont dû en supporter angoisses et indisponibilité. Qu'ils soient eux aussi remerciés.

Le but de ce travail est, comme le titre l'indique, de s'interroger sur l'efficacité de la programmation logique par contraintes dans le domaine de la conception d'horaires et, singulièrement, de l'horaire d'une session d'examens. Il paraît inconcevable, au cours d'une telle réflexion, de ne s'intéresser qu'à des expérimentations en chambre, d'autant que l'expérimentateur a besoin, pour mener à bien sa tâche, de s'imprégner de l'expérience des personnes qui conçoivent ces horaires manuellement. Cette imprégnation n'est possible qu'au terme de discussions parfois longues. Ces concepteurs se mettent alors à espérer la réalisation d'un produit très concret. D'où la nécessité d'un développement de bonne qualité et pas d'un simple prototype. Il faut en effet un programme qui fonctionne, qui donne de bons résultats[1] et qui offre une interface de communication élémentaire, agréable, souple, transparente.

Il y a donc une sorte de tension, dans ce genre de tâche, entre l'envie de prendre du temps pour tester divers modèles, diverses stratégies et celle de développer un produit qui, pour ne pas être rejeté, devra faire montre de qualités d'efficacité et d'utilisabilité.

Nos objectifs se sont donc répartis dans plusieurs directions:

- modéliser,
- expérimenter,
- développer une application robuste et efficace,
- développer une interface graphique de qualité…

À cela s'ajoute un objectif plus personnel, celui de maîtriser un paradigme de programmation que nous avions, jusque là, peu expérimenté.

En réalisant ce travail, nous avons tenté de satisfaire tous ces objectifs et nous avons pensé à tous ceux qui ont des attentes particulières dans le domaine: à ceux qui se sont déjà frottés à un tel problème et qui voudraient connaître ou explorer d'autres pistes que celles qu'ils ont suivies, à ceux qui souffrent épisodiquement de devoir le résoudre et à nous-mêmes, égoïstement, qui avions envie d'explorer un paradigme de programmation dont nous n'avions pas l'expérience.

Dans le **chapitre 1**, nous parlerons de ce mythe que constitue, quelque part, la constitution d'un « bon » horaire par une machine. Nous dirons un mot des méthodes qui ont connu un certain succès et nous tenterons de justifier le choix de l'approche *CLP* (programmation logique par contraintes) que nous avons choisie en expliquant ses principes les plus généraux. Nous parlerons aussi de ce que nous avons appelé les niveaux de modélisation et expliquerons pour quel niveau nous avons opté et pourquoi.

---

[1] …au moins aussi bons que ceux que le concepteur peut construire

Le **chapitre 2** est consacré à la modélisation. Le choix de l'approche étant réalisé, il s'agit de s'intéresser à tout ce qui, dans le contexte examiné, fait l'objet d'une modélisation possible. Nous verrons en quoi la *CLP* nous facilite la tâche. Nous développerons une modélisation des données, des relations entre ces données et enfin des contraintes avant d'expliquer quelles simplifications nous avons effectuées à des fins de réalisation rapide et concrète d'un logiciel fournissant des résultats acceptables.

Nous détaillons dans le **chapitre 3** la structure du programme et nous illustrons la manière d'implanter la création des variables, la définition des contraintes et les techniques de labélisation. Cette partie est destinée à ceux qui connaissent la programmation logique et qui sont intéressés de voir comment très concrètement, cette implantation peut avoir lieu.

Le **chapitre 4** décrit un certain nombre d'expérimentations menées et les résultats qu'elles ont produits.

Dans le **chapitre 5**, l'interface de communication est décrite avec les possibilités qu'elle offre. Nous y détaillons comment cette interface permet la conversion des données fournies via des formulaires de pages web en données traitables par le programme *Prolog*.

Enfin, le **chapitre 6** ouvre quelques perspectives d'amélioration du programme et envisage même la possibilité d'un autre type de développement avec ses avantages et ses inconvénients.

Chapitre 1 PROBLÉMATIQUE ET CHOIX D'UNE APPROCHE

Ce qui est le meilleur dans le nouveau est ce qui répond à un désir ancien.
(Paul Valéry)

1.1 L'homme contre la machine[2]

1.1.1 La numérisation

Informatique et numérisation ne vont pas l'un sans l'autre. Une bonne manière de faire percevoir à quelqu'un le profit qu'il peut tirer d'un ordinateur et, plus encore, d'un programme que celui-ci exécute, est de lui faire comprendre que cette numérisation est, à la fois, ce qui va permettre d'automatiser une foule de tâches et, dans le même temps, ce qui va générer des sources potentielles de problèmes insoupçonnés de par le côté inhumain et dépourvu de sens des traitements effectués. Les traitements d'information qui étaient effectués par l'Homme sont, depuis l'apparition du traitement automatique, regardés au travers du filtre de la numérisation. Une tâche étant donnée, les questions qui se posent sont: cette numérisation est-elle possible? partiellement possible? impossible[3]? Les tâches qui génèrent le plus de problèmes sont celles qui mobilisent chez l'être humain une foule de compétences diverses mais aussi celles qui admettent une très grande quantité de solutions pratiquement impossibles à classer sur une échelle de qualité. Si résumer un texte demande de la part de celui qui réalise le travail, une bonne connaissance de la langue, de l'orthographe, de la grammaire, ainsi que des capacités de compréhension, de repérage des idées importantes, etc., c'est aussi un problème qui, pour un texte donné, admet un ensemble quasi infini de solutions, certaines pouvant être jugées meilleures que d'autres, sans que l'on puisse déterminer facilement pourquoi une solution est meilleure qu'une autre.

1.1.2 De la difficulté de formaliser

La constitution d'un horaire est une tâche qui, sans atteindre la complexité de l'activité de synthèse dont nous venons de parler, s'y apparente un peu. Tous ceux qui s'y attèlent manuellement tiennent compte d'une foule de données, qui sont souvent de l'ordre du détail, sans qu'ils en aient eux-mêmes toujours bien conscience[4]. Le concepteur d'un horaire doit souvent juger de la pertinence d'une contrainte et ce jugement, voilà bien quelque chose de difficile à spécifier[5]. Mais de la même façon qu'un juge ne peut se contenter de la loi et fait aussi appel à la jurisprudence, il semble bien que quels que soit les critères choisis, ceux-ci pourront toujours avoir pour effet de repousser des solutions qui auraient pu être acceptées sans éliminer certaines solutions pourtant difficilement acceptables. C'est précisément cet aspect des choses qui caractérise la difficulté de constitution automatique d'un horaire. D'une part, le traitement est fastidieux. Idéalement, il faut essayer de nombreuses solutions mais sans doute pas toutes les solutions. Cette facette du problème nous encourage à passer à l'automatisation en

---

[2] De manière assez amusante, c'est ainsi qu'inévitablement le concepteur manuel regarde l'informaticien s'approcher de son problème.

[3] ...à l'heure actuelle, bien entendu.

[4] Il faut parfois plusieurs heures de discussion avec les concepteurs d'horaires pour qu'ils vous livrent certains "secrets de fabrication" sans qu'il n'y ait eu vraiment volonté de leur part de cacher ces informations.

[5] Dans le cas particulier d'un horaire de session d'examens, s'il n'est pas très raisonnable de proposer deux épreuves à un étudiant le même jour, faut-il pour autant en faire une règle à laquelle on ne peut déroger? Et lorsque la dérogation est possible, sur base de quels critères doit-elle se faire? Lorsqu'il s'agit d'épreuves plus légères? lorsque l'étudiant dispose d'un long temps de blocus?...

confiant aux exécutants infatigables que sont les ordinateurs programmés, le soin de calculer et/ou de rechercher. D'autre part, il semble très difficile de couler dans un moule toute cette jurisprudence qui entoure un problème donné dans un contexte spécifique. Cette difficulté à formaliser ces capacités de jugement fait que les adversaires de l'automatisation considèrent, et ils ont sans doute partiellement raison, que les horaires constitués par les machines sont dépourvus d'humanité. Ajoutons encore que pour un automate à la recherche d'une solution dans le respect d'un certain nombre de règles, tout ce qui n'est pas interdit est nécessairement autorisé. Le concepteur est donc plus ou moins tenu à l'exhaustivité dans la description des contraintes, ce qui n'est pas toujours réalisable ou raisonnablement imaginable.

1.2 Le choix d'une approche

1.2.1 La coloration des graphes

L'intérêt pour une informatisation des conceptions d'horaires est presque aussi vieux que l'informatique. Il y a une vingtaine d'année, ce type de problème était volontiers transformé en problème de coloration des graphes donnant lieu au développement d'algorithmes devenus célèbres.

Rappelons brièvement et simplement son principe. Il se base sur le fait que la détermination d'un horaire consomme des ressources et que celles-ci ne sont pas inépuisables. Citons au rang de ces ressources: les locaux, mais aussi les enseignants et les classes[6] d'étudiants. Dès lors qu'une période de l'horaire mobilise une partie de ces ressources, il ne faut pas que celles-ci soient mobilisées par ailleurs au même moment. Les périodes sont les sommets d'un graphe, chacun d'eux étant relié par un arc (une arête si le sens de la relation n'a pas d'importance) à ceux qui mobilisent au moins une ressource identique (le professeur, la classe, le local). Le but du jeu est de colorer chacun des sommets de sorte que deux sommets d'une même couleur ne soient pas en relation, en utilisant un nombre de couleurs inférieur au nombre de périodes disponibles.

Prenons l'exemple très théorique d'un établissement dans lequel il y a:

- deux professeurs (*p1* et *p2*),
- quatre cours dispensés (*c1*, *c2*, *c3* et *c4*),
- cinq étudiants (*e1* et *e2* dans la classe *cl1*et *e3*, *e4* et *e5* dans la classe *cl2*).

Le fait que le professeur *p1* examine pour le cours *c1*, les étudiants *e1* et *e2* est modélisé par l'existence d'un sommet

$$x1\ (p1, c1, e1, e2)$$

En imaginant que *p1* enseigne aussi le cours *c3* aux étudiants de la classe *cl2*, un autre sommet est

$$x2\ (p1, c3, e3, e4, e5)$$

Ces deux sommets possèdent une ressource commune *p1*. Ils sont donc reliés par une arête du graphe. Dès lors, ils doivent être colorés différemment. Entendez par là que des moments d'examen différents doivent leur être attribués.

Si on ajoute encore que:

- *p2* interroge sur *c2* les étudiants *e3* et *e4* de la classe *cl2* (sommet *x3*)
- *p2* interroge sur *c4* l'étudiant *e3* (sommet *x4*)

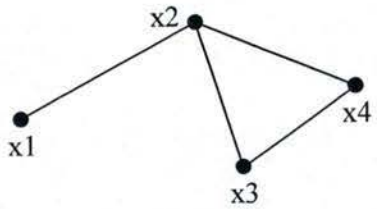
on obtient un graphe qui ressemble au suivant[7].

---

[6] Si à l'époque, les concepts de classe et d'année d'étude étaient encore bien solides, ils ont aujourd'hui pris du plomb dans l'aile ou, en tous cas, ils ont considérablement évolués.

[7] Nous avons considéré, dans cet exemple simple, qu'un cours ne donnait lieu qu'à une seule épreuve, ce qui n'est pas toujours le cas (épreuve écrite ≠ épreuve orale).

On s'aperçoit rapidement que trois couleurs sont nécessaires si on souhaite que les sommets adjacents soient de couleurs différentes.



Autour de cette idée simple, peut se développer une théorie qui consiste à examiner les stratégies possibles de coloration pour éviter un gaspillage des couleurs. Les algorithmes de coloration des graphes permettent d'aborder la tâche avec différentes stratégies.

Dans le cas particulier de la constitution d'un horaire de session d'examens, la multiplication des étudiants ayant un statut particulier (inscription sur deux années) et des cours à option ne partitionnant pas vraiment, au sens mathématique du terme, l'ensemble des étudiants, nous pousse à considérer que la notion de classe (ou d'année d'étude) disparaît quelque peu. Cette évolution ne semble pas devoir faire marche arrière, loin de là. Pour cette raison, il paraît intéressant, comme dans le petit exemple qui a précédé, de considérer chaque étudiant comme une ressource plutôt que l'année d'étude dans laquelle il a pris son inscription principale.

Ces dernières considérations et la multiplication de contraintes locales relativement variables[8] nous incitent à tenter une autre voie.

1.2.2 La programmation logique par contraintes

La programmation logique est bien connue de ceux qui s'intéressent aux problèmes liés à des contextes dans lesquels les bases de connaissances sont importantes, et dans lesquels des règles peuvent être énoncées afin de permettre la génération de connaissances nouvelles. Cette génération est rendue possible sous forme d'interrogation d'un système dans lequel la programmation est essentiellement déclarative et les programmes, constitués de **faits** et de **règles**. À l'instar de *SQL*, considéré comme une sorte de langage générique d'interrogation de bases de données, *Prolog* est un langage générique définissant les bases d'un langage de programmation logique.

La programmation logique fonctionne essentiellement sur base d'un algorithme d'unification logique permettant la production de solutions en réponse à des requêtes. L'addition d'un solveur de contraintes permet d'étendre le discours. Le plus souvent, les contraintes apparaissent dans le programme mais elles peuvent également apparaître dans une requête voire dans les réponses fournies.

Le solveur peut utiliser différents algorithmes de « satisfaction de contraintes » basés sur des techniques variées dont un des buts est de restreindre l'espace de recherche. On sait en effet que l'explosion combinatoire est importante dans ce type de problème. Nous décrivons certaines de ces techniques dans un des paragraphes qui suivent.

Une famille de langages s'est développée autour du schéma **CLP** (*Constraint Logic Programming*), les membres de cette famille se spécialisant en s'intéressant à des domaines particuliers dans lesquels les contraintes peuvent être évaluées:

- *CLP(R)* les réels,
- *CLP(Q)* les rationnels,
- *CLP(FD)* les domaines finis (*Finite Domains*),

---

[8] Si le principe de la coloration des graphes est remarquable de simplicité, la complexité nous paraît résider dans la modélisation des contraintes, ce qui pourrait décourager une modélisation coûteuse de certaines contraintes considérées comme non fondamentales mais pourtant garantes de la qualité d'un horaire.

- *CLP(FT)* les arbres finis (*Finite Trees*),
- ...

Les programmes écrits dans ces langages sont donc constitués d'instructions *Prolog* classiques et d'instructions de description des contraintes destinées à un solveur de contraintes auquel *Prolog* fait appel.

La direction choisie ici est celle de *CLP(FD)* c-à-d. celle de la programmation logique par contraintes utilisant des variables de domaines finis.

1.2.3 Variables FD et contraintes

En *CLP(FD)*, on ne parle plus de sommets mais de variables dont les valeurs sont à déterminer.

À titre d'exemple, dans le cas précis d'un horaire de session, il est clair qu'une des variables est, par exemple, la demi-journée[9] *X1* pendant laquelle le professeur *P1* fera passer l'examen écrit du cours *C1*. Une autre variable est la demi-journée *X2* pendant laquelle l'étudiant *E* passera son examen oral pour le cours *C2*. Dans une solution finale, *X1* et *X2* devront prendre une valeur précise, valeur correspondant à une demi-journée. Le nombre de valeurs possibles au départ pour *X1* et *X2* est évidemment limité, la session n'étant pas extensible. On peut donc facilement numéroter ces valeurs. En ce sens, on dit que le domaine des variables est fini. De même, le nombre de groupes à constituer pour un examen oral est un nombre entier limité. Il peut donc également être considéré comme une variable de domaine fini.

La **variable de domaine fini** (nous dirons désormais variable FD) est donc un peu le pendant de la couleur dans l'approche par coloration des graphes. Nous noterons cependant que si dans cette dernière approche, le sommet comprend l'ensemble des étudiants présentant l'épreuve concernée et le professeur qui interroge, dans l'approche *CLP*, la variable concerne soit un étudiant, soit un professeur. Le fait de leur présence simultanée à certains moments et à certains endroits se traduira par des **contraintes** mettant en jeu ces variables.

Si *Y1* est la demi-journée pendant laquelle le professeur *P2* interroge oralement le groupe[10] de l'étudiant *E* dans le cours *C2*, il faudra nécessairement que

$$X2 = Y1$$

D'autres exemples feront intervenir d'autres opérateurs entre les variables. Ainsi, si le professeur *P1* interroge oralement lors du demi-jour *Y2* un groupe d'étudiant sur le cours *C1* et que l'on considère qu'un écrit d'une épreuve doit précéder l'oral, on aura obligatoirement

$$X1 < Y2$$

Bien entendu, au niveau d'un programme, les contraintes sont exprimées de manière plus générale, mais ces exemples permettent de planter le décor.

1.2.4 La satisfaction des contraintes et le rôle des solveurs

La tâche des solveurs de contraintes est, en quelque sorte, de gérer les domaines des variables qui sont à déterminer. Le nombre pléthorique de solutions à essayer est tel qu'on ne peut espérer, même avec des ordinateurs puissants, obtenir de bons résultats dans des délais convenables si on ne développe pas quelques stratégies opératoires. Ces stratégies existent à au moins trois niveaux:

- au niveau du programmeur,
- au niveau des primitives du langage,
- au niveau de la gestion des domaines de valeurs.

---

[9] ...comprenez chaque fois: "la valeur de la demi-journée" dès lors qu'elles peuvent être numérotées

[10] Un professeur interroge oralement plusieurs étudiants sur une demi-journée. C'est à l'ensemble de ces étudiants que fait ici référence le mot "groupe".

Le programmeur peut décider d'orienter l'ordre dans lequel les variables sont examinées. On parle de la manière d'organiser le **labeling** des variables. Dans le cas qui nous occupe, on peut choisir de fixer d'abord les valeurs des variables qui concernent les professeurs avant celles qui concernent les variables des étudiants. On peut décider, comme le ferait d'ailleurs un concepteur manuellement, de « placer » d'abord dans l'horaire les épreuves écrites avant les épreuves orales.

De même, après avoir choisi la variable dont la valeur est à fixer, comment choisir parmi les valeurs encore disponibles? Le solveur peut choisir la plus petite, la plus grande des valeurs, choisir aléatoirement,... On parle d'**heuristiques** et d'**algorithmes stochastiques** pour désigner le traitement qui s'effectue à ce niveau de manière transparente pour le programmeur.

Il s'avère également fondamental de pouvoir rétrécir rapidement le domaine des variables, soit pour aboutir sans tarder à une solution, soit pour se rendre compte le plus vite possible que la voie choisie n'est pas la bonne. Diverses techniques existent parmi lesquelles, les **techniques de consistance** que nous détaillons un peu dans le paragraphe suivant.

1.2.5 Les techniques de consistance

Les contraintes mettent en jeu un certain nombre de variables. On peut s'intéresser plus particulièrement aux contraintes unaires et binaires.

Les contraintes unaires permettent d'agir immédiatement sur le domaine de la variable concernée.

$$X :: 1..5 \wedge X < 3 \Rightarrow X :: 1..2$$

Traduisez: la variable X peut prendre une valeur entre 1 et 5 et la variable X doit prendre une valeur inférieure à 3.

Les contraintes n-aires pouvant se ramener à un ensemble de contraintes binaires, ces dernières sont intéressantes car elles permettent d'établir un graphe de contraintes dans lequel les algorithmes peuvent exploiter les techniques éprouvées de recherche.

Les techniques classiques, consistant par exemple à générer une solution avant de la tester (**generate-and-test**), ainsi que les méthodes classiques utilisant le **backtracking** détectent souvent fort tard les problèmes. Pour ces raisons, des techniques ont été mises au point pour repérer le plus tôt possible une éventuelle inconsistance. De bons résultats sont obtenus en utilisant la **consistance aux nœuds** et la **consistance aux arcs**[11].

Les nœuds du graphe représentent les variables et les arcs (ou les arêtes), les contraintes binaires. On a une solution au problème posé lorsqu'on dispose d'une valeur pour chacune des variables et que ces valeurs satisfont les contraintes.

La consistance aux nœuds

C'est la consistance simple résultant des contraintes unaires. L'inconsistance peut être aisément résolue en enlevant du domaine de la variable, les valeurs qui ne satisfont pas cette contrainte unaire.

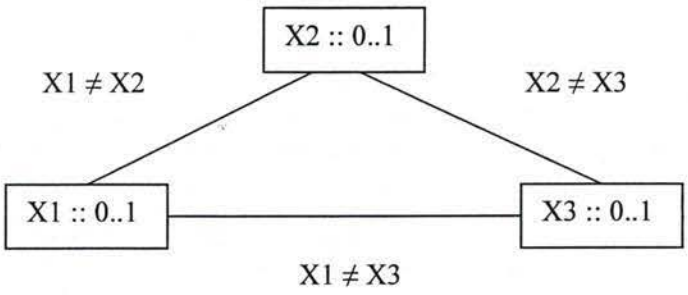
La consistance aux arcs

Si le graphe des contraintes est consistant aux nœuds, on peut s'intéresser à la consistance aux arcs puisque ceux-ci traduisent les contraintes binaires. La consistance aux arcs se traduit par le fait que pour un arc *(X, Y)*, à toute valeur de *X* correspond au moins une valeur de *Y* telles que *X* et *Y* vérifient la contrainte. Cette technique permet de restreindre l'ensemble des valeurs des variables mais n'assure pas que les valeurs choisies au hasard dans chaque domaine de variable constituent une solution sauf

---

[11] Il est possible de définir une consistance qui va au-delà d'un seul arc (K-consistency, path consistency) mais le maintien de ces dernières se révèle assez coûteux. A la limite, le maintien d'une consistance totale éliminerait toute recherche mais à quel prix. Il importe donc de choisir le bon niveau de consistance pour éviter un coût trop élevé de son maintien mais qui élague suffisamment l'arbre de recherche pour éviter que celle-ci ne prenne un temps infini.

dans le cas très particulier où chaque domaine se réduit à une seule valeur. Voici un exemple élémentaire montrant un graphe consistant aux arcs n'admettant pas de solution.



Chaque arc est consistant puisque pour chaque valeur de chaque variable, il existe une valeur de l'autre variable qui soit différente. On a pourtant trop peu de valeurs à attribuer aux trois variables pour qu'elles soient toutes différentes.

On trouve plusieurs algorithmes qui assurent ces deux types de consistance dans [17] ainsi qu'une discussion sur leur efficience.

La propagation des contraintes

On peut considérer que le backtracking utilise, d'une certaine manière, une technique de consistance dans la mesure où la vérification de cette consistance se limite aux variables dont les valeurs ont déjà été fixées. C'est évidemment déjà beaucoup mieux que de se donner un jeu de valeurs pour chacune des variables et de vérifier si ce jeu convient (ce que fait le *GT*).

Mais pour aller plus loin, il est possible et intéressant de combiner les méthodes classiques avec les techniques qui viennent d'être brièvement décrites en encapsulant un algorithme de consistance dans un algorithme de backtracking. Dès qu'une valeur est fixée, l'algorithme peut vérifier la consistance en ne s'intéressant qu'aux arcs liés au nœud constitué par la variable. S'il arrive qu'un domaine soit complètement réduit, l'algorithme de backtracking fait alors son travail.

Parmi les techniques possibles, citons le **Forward Checking** et le **Look Ahead**.

Le *Forward Checking* s'intéresse à la variable courante et aux conflits qui pourraient se produire avec chacune de celles qui vont être examinées à sa suite. Cette technique permet d'élaguer rapidement certaines branches de l'arbre de recherche.

Le *Look Ahead* s'intéresse en plus aux conflits qui pourraient se produire entre les futures variables elles-mêmes. En d'autres termes, la technique garantit le maintient de la consistance aux arcs[12].

L'intérêt de ces méthodes, c'est que lorsqu'une nouvelle variable est examinée, la consistance avec les anciennes est garantie et les vérifications pour ces variables ne sont plus nécessaires.

Le solveur associé au langage que nous utilisons pour la conception des horaires utilise, selon la syntaxe des opérateurs choisis, une consistance aux arcs totale ou partielle.

1.2.6 CLP(FD) vs coloration des graphes

Un parallèle avec la coloration des graphes peut être fait. Si les algorithmes de coloration doivent se contenter d'un nombre limité de couleurs, il en est de même du solveur de contraintes qui doit composer avec un nombre fini de valeurs possibles pour les variables définies dans le programme. Au départ, les variables à déterminer disposent d'un ensemble de valeurs possibles, l'équivalent d'un nombre de couleurs pour les sommets.

---

[12] Le problème des huit reines sur un échiquier est bien connu. Ramené à quatre reines sur un échiquier de seize cases, le backtracking conduit à une réussite après 18 échecs, le Forward Checking à une réussite après 2 échecs et le Look Ahead à une réussite après un seul échec.

Prenons l'exemple de l'horaire de session d'examen. Dans une approche liée à la coloration des graphes, on pourrait considérer que le professeur *p* interrogeant les étudiants de l'année *a* pour l'écrit du cours *c* définit un sommet et que tout autre sommet faisant intervenir *p* ou *a* dans une autre épreuve ne peut avoir la même couleur que le sommet précédent[13]. Le nombre de couleurs disponibles est celui des demi-jours de session ouvrables.

Dans l'approche *CLP(FD)*, il importe de déterminer ce que sont les variables. En l'occurrence, il en existe une pour chaque couple (*étudiant*, *épreuve_à_passer*) et une pour chaque couple (*professeur*, *épreuve_à_faire _passer*)[14]. Les valeurs que doivent prendre ces variables sont à choisir parmi le nombre fini de valeurs correspondant aux demi-jours de session.

Dans un formalisme assez proche de ce qui se fait dans le domaine de la programmation logique, on pourrait écrire:

passe(nom_étudiant, cours, type_de_cours, X)

pour traduire le **fait**[15] que l'étudiant *nom_étudiant* doit passer l'épreuve *type_de_cours* du cours *cours* au moment *X*. Ce moment correspond à un demi-jour et possède donc un ensemble de valeurs possibles en cours d'exécution du programme et une valeur fixée lorsque celui-ci fournit une solution.

On pourrait aussi écrire:

examine(nom_professeur, cours, type_de_cours, Y)

pour signaler que le professeur *nom_professeur* interroge pour l'épreuve *type_de_cours* du cours *cours* au moment *Y*.

Dans ce cas, *X* et *Y* prennent leurs valeurs dans le même intervalle, du moins au départ puisque le professeur peut mentionner qu'il est indisponible à certaines dates.

Les contraintes agissent de deux manières, soit en restreignant immédiatement l'ensemble des valeurs possibles de certaines variables, soit en mémorisant des relations qui lient les variables entre elles et qui deviennent déterminantes lorsque l'une ou l'autre de ces variables est fixée.

Par exemple, les valeurs correspondant aux dimanches ne sont pas acceptables. Des contraintes le signalent et le solveur rétrécit le domaine des variables[16]. Si on considère comme inacceptable qu'un étudiant passe deux examens le même jour, une relation dira que les valeurs des variables correspondant à un même étudiant doivent différer de deux unités (demi-jours). Une telle contrainte ne rétrécit pas directement le domaine des variables mais le fera lorsqu'il s'agira de commencer à fixer les valeurs pour certaines variables[17].

Dans l'une comme dans l'autre approche, on devine que l'ordre de détermination des valeurs (couleurs) n'est pas sans répercussion sur la (les) solution(s) finale(s). Dans le cas spécifique de *CLP(FD)*, cette opération doit être programmée après la prise en compte des contraintes. Elle porte le nom de **labeling**.

---

[13] Le problème du local est évacué ici. Nous verrons par la suite qu'il n'est pas crucial pour un horaire de session d'examens.

[14] Il n'y a pas, à proprement parler, de bijection entre l'ensembles des épreuves que doit passer un étudiant *e* pour un cours *c* (généralement un écrit et/ou un oral) et l'ensemble des épreuves à faire passer pour ce même cours *c* par le professeur *p* qui l'enseigne (puisqu'un oral se répartit souvent sur plusieurs demi-journées).

[15] Il s'agit bien d'un fait comme on l'entend en programmation logique à la différence près qu'à la variable X est associé un ensemble de valeurs et non une valeur unique.

[16] Dans ce cas particulier, les domaines de toutes les variables sont restreints de la même manière.

[17] Il faut bien commencer quelque part!

1.2.7 Pourquoi CLP(FD)?

De ce qui précède, nous pouvons retenir que le nombre de variables à prendre en compte est assez élevé si nous admettons que le concept de classe est dépassé. Nous ne pouvons évidemment présumer de ce que fournirait comme solution un algorithme de coloration de graphe qui considérerait comme sommets des n-uplets du style

(professeur, cours, type_d_épreuve, étudiant_1, étudiant_2,..., étudiant_m)

où

étudiant_1, étudiant_2,..., étudiant_m

sont les étudiants qui doivent présenter l'épreuve concernée puisque nous n'avons rien expérimenté à ce propos. On remarque cependant que dans l'approche *CLP(FD)*, des couples suffisent à déterminer les variables puisque ce sont des contraintes qui s'occuperont de lier les professeurs aux étudiants par l'intermédiaire des cours.

Concevoir qu'un horaire soit une solution satisfaisant à un ensemble de contraintes est une idée aisément admissible et plutôt naturelle[18]. La difficulté est certainement de les formaliser correctement. D'autre part, la programmation logique semble bien s'accommoder de la recherche de solutions par des méthodes basées sur les essais et les erreurs puisque celle-ci procède par des techniques de backtracking pour l'élaboration des solutions. Il faudra cependant veiller à ce que ce backtracking soit « allégé ».

On devine que la difficulté se situe davantage du côté de la manière dont la recherche de la solution va être menée pour obtenir des solutions qui soient intéressantes, étant donné que le nombre de possibilités à envisager est énorme (pour ne pas dire infini à l'échelle de l'être humain). L'augmentation de la capacité des mémoires des ordinateurs et de la vitesse de travail des microprocesseurs ne suffisent hélas pas à explorer toutes ces solutions dans des délais acceptables.

Le mariage entre la programmation logique et des solveurs de contraintes dans la *CLP* donne donc pas mal d'espoir en matière d'élaboration d'horaires. Nous tâcherons de vérifier si ces espoirs sont raisonnables.

Nous avons utilisé pour nos expériences la **version 1.2.16** de **GNU-Prolog**[19] qui est un compilateur du domaine libre mis au point par Daniel Diaz et qui inclut les éléments nécessaires à la résolution des contraintes sur des domaines finis .

1.3 Plusieurs niveaux de modélisation

1.3.1 De la difficulté d'une modélisation très générale

Avant de nous intéresser à la modélisation, nous pouvons nous poser la question de savoir s'il est possible d'examiner le problème de la constitution d'un horaire en toute généralité. Pour cela, il faudrait tâcher d'isoler un certain nombre de concepts communs à tous les types d'horaires. Le travail risque d'être difficile.

Les problèmes de constitution d'horaire peuvent être de natures très différentes[20].

---

[18] ...en ce sens que l'expression d'une contrainte est plus explicite qu'elle le serait dans une approche de type "coloration de graphes"

[19] http://gnu-prolog.inria.fr/

[20] Il est remarquable de constater que de nombreux papiers sont consacrés à la planification des cours dans le cadre particulier de l'université. Généralement, leurs auteurs reconnaissent prendre en compte des contraintes qui sont très locales en introduisant des paramètres spécifiques à leur propre université et, de surcroît, ils limitent généralement le champ d'implantation de leurs solutions à un seul département ou à une seule faculté.

Les contraintes liées à l'établissement d'un horaire hebdomadaire n'ont pratiquement rien à voir avec celles qui s'imposent dans l'établissement de l'horaire d'une session d'examen. Il est difficile, par exemple, de dégager des classes de contraintes communes. Dans le premier cas, ce sont des années d'études qui sont concernées et, pour une année donnée, les périodes disponibles pour l'horaire seront probablement majoritairement occupées. Dans le second, si le nombre d'épreuves orales est important, ce sont plutôt les étudiants qui sont concernés. Même si, pour un étudiant donné, le nombre de périodes éligibles est largement trop élevé, il est évident que la difficulté de constituer l'horaire provient essentiellement du nombre important d'étudiants.

Une modélisation de très haut niveau n'est donc pas particulièrement évidente d'autant que, comme le soulignent certains auteurs, l'organisation des cours de tous niveaux a subi de profondes mutations.

En Belgique, les enseignements secondaire et universitaire qui étaient clairement organisés en années d'étude et en sections voici quelques dizaines d'années évoluent aujourd'hui vers un enseignement qui sans devenir modulaire, offre à l'étudiant un large éventail de cours à option à l'intérieur d'une même année d'étude, ou plus encore, lui permet de suivre des cours et de présenter des épreuves qui concernent des années différentes.

Ce phénomène tendant à se généraliser, on voit mal comment les futurs programmes de constitution d'horaires pourraient éviter de se baser sur une modélisation qui tient compte de l'étudiant plutôt que d'un groupe « classe ». Selon les contextes, on peut envisager de constituer l'horaire en groupant les étudiants en années d'étude, puis adapter ce dernier aux cas particuliers que constituent les étudiants « à cheval » sur deux ou plusieurs années. Il semble toutefois plus raisonnable de généraliser le problème en oubliant la notion d'année et en considérant que les étudiants sont inscrits à des modules pour lesquels ils doivent subir les épreuves d'examens. Il est évident que cette seconde option entraîne une plus grande complexité du problème dans la mesure où le regroupement des étudiants s'effectue cette fois module par module et non sur un ensemble de modules.

Un autre obstacle à la modélisation de haut niveau, c'est que l'on dispose généralement de solutions antérieures, réalisées manuellement et qui semblent convenir aux individus concernés. Craignant sans doute qu'une modélisation trop éloignée de la réalité ne provoque de grands bouleversements, certains ont envisagé de construire un horaire par adaptations successives d'une solution existante, tenant compte notamment des changements annuels dans les attributions et les desiderata des enseignants. On trouve la description d'une telle stratégie dans [3].

On perçoit encore un obstacle à une modélisation qui soit très générale: la spécificité même du domaine auquel le problème s'applique (en l'occurrence en ce qui nous concerne, les cours et les examens à l'université) et sa capacité à évoluer dans le temps.

La lecture de nombreux rapports sur le sujet nous conforte dans l'idée qu'il y a presque autant de modèles que de problèmes à résoudre. Chaque problème est examiné avec ses spécificités qui apparaissent très tôt dans la modélisation. À première vue, on a l'impression que chaque effort de modélisation réalisé dans un contexte précis, n'a que fort peu de chance de trouver d'autres terrains d'application et d'offrir ainsi de réaliser des économies d'échelles.

Cela dit, des tentatives de modélisation sont effectuées à différents niveaux.

1.3.2 Des modélisations de haut niveau

Malgré ce qui vient d'être dit, on trouve des modélisations de haut niveau, notamment dans [1] l'**approche hiérarchique** que nous examinons dans le chapitre 6 sur les perspectives. Il s'agit d'une approche très théorique et très fondée qui peut, à priori, se greffer sur une assez large gamme de problèmes de planification. Très simplement dit, l'idée centrale est que les contraintes liées à l'établissement d'un horaire ne sont pas toutes d'une même importance. Dès lors, un solveur de contraintes bien construit peut tenir compte d'une certaine hiérarchie de celles-ci et choisir de respecter celles qui optimisent une certaine fonction. C'est une approche très séduisante de premier abord. Nous verrons qu'elle ne résout pas tous les problèmes.

Dans le même ordre d'idée, on trouve en [3] une autre approche intéressante dans la technique de **propagation des contraintes douces**. Dans un système sur-contraint, on admet que des contraintes

qualifiées de « douces » ne puissent être toutes satisfaites. Dès lors, au cours de la résolution, celles qui ne le sont pas augmentent en proportion de leur poids la valeur d'une fonction objective que le solveur doit minimiser.

1.3.3 Des modélisations de bas niveau

Dans la pratique, il arrive souvent que les solutions apportées aux problèmes de constitution d'horaire soient de l'ordre de la confection sur mesure plutôt que du prêt-à-porter. Cela signifie que le choix de la modélisation est très proche du problème et n'est pas envisagée dans un contexte plus large. Dans ces recherches de solutions, on évite de travailler avec des contraintes non absolument requises. On trouvera en [2] une étude de ce type. Il s'agit de l'établissement de l'horaire hebdomadaire des deux années d'études d'un seul département. On constatera que le nombre de variables entrant en jeu est relativement peu élevé et que les contraintes y sont très spécifiques.

Avec cette façon de procéder, on investit peu dans la modélisation puisqu'on se contente de traduire de manière assez directe des contraintes plutôt incontestables. Cependant, les résultats produits par ces modélisations plus élémentaires risquent de garder les traces d'une économie réalisée en évitant, par exemple, l'utilisation de contraintes douces. En négligeant les contraintes pas vraiment nécessaires, on risque aussi d'obtenir des solutions moins élégantes, bien que respectant les contraintes imposées.

En ce qui nous concerne, nous avons choisi une modélisation de ce type. Elle se justifie par le nombre assez limité de contraintes à prendre en compte et par le peu d'expérience de solutions apportées jusqu'à présent au problème envisagé.

Chapitre 2 MODÉLISATION

Ce qui est créé par l'esprit est plus vivant que la matière.
(Charles Baudelaire)

2.1 Modélisation d'une session d'examen à l'université

2.1.1 Une classification très générale

Malgré les commentaires quelque peu pessimistes de la fin du chapitre dernier sur les tentatives de modélisations générales, nous pouvons tenter de dégager quelques points communs aux différentes classes de problèmes d'horaire.

Dans l'approche que nous avons choisie, la notion de contrainte est centrale. Aussi nous paraît-il intéressant de caractériser ces contraintes d'au moins trois manières.

Certaines contraintes sont requises alors que d'autres peuvent être de l'ordre du souhait.

Toutes les contraintes ne sont pas de même importance. Certaines traduisent des souhaits.

- Il est souhaitable que les étudiants ne passent pas deux épreuves le même jour.
- Il est souhaitable que les oraux d'un même cours ne soient pas dispersés dans l'horaire.
- ...

Certaines approches considèrent que ces contraintes sont des contraintes douces, ou qu'il faut pondérer, voire hiérarchiser pour obtenir des solutions qui soient plus humaines, plus acceptables. Une contrainte douce, par exemple, ne doit pas nécessairement être rencontrée dans la solution. De ce fait, elle n'est pas bloquante. Les contraintes douces sont considérées dans leur globalité, ce qui permet à un solveur d'optimiser une fonction objective en en respectant certaines et en en négligeant d'autres.

D'autres contraintes en revanche sont requises. Il s'agit, soit de contraintes qui ne sont pas propres au contexte et qui sont parfois appelées pour cette raison « contraintes indirectes », soit de contraintes considérées comme importantes d'un point de vue pédagogique.

- Un étudiant ne peut se présenter à deux examens en même temps (contrainte indirecte).
- L'épreuve écrite d'un cours doit se dérouler avant les épreuves orales du même cours qui peuvent avoir un objectif de rattrapage (contrainte pédagogique).
- ...

Notez qu'une contrainte pédagogique n'est pas forcément une contrainte requise[21].

Les contraintes indirectes correspondent à des limitations dans les ressources.

Un local ne peut être occupé par un groupe trop important ou par deux groupes en même temps[22]. Les professeurs et les étudiants n'ont pas le don d'ubiquité. Du point de vue de la modélisation, étudiants comme professeurs peuvent être considérés comme des ressources en exemplaire unique.

---

[21] Il arrive qu'un professeur ne considère pas comme important le fait que l'écrit précède l'oral. Dans un contexte de programmation où l'on n'accepte que des contraintes requises, il faut composer avec de telles situations.

[22] ...à moins qu'il ne s'agisse de réaliser des économies de surveillants.

Les contraintes pédagogiques sont propres au domaine d'application et au contexte.

Concevoir un horaire de session d'examens n'est pas le même travail que de concevoir un horaire de cours hebdomadaire. Les contraintes qui caractérisent ces horaires n'ont en général pas grand-chose en commun. Si la prise en compte des locaux est peu importante dans le premier cas, elle est fondamentale dans le second. La densité d'occupation est en effet beaucoup plus élevée. De plus, des contraintes d'utilisation de locaux spécialisés peuvent s'ajouter. Peut-on même parler de contraintes d'organisation d'une session d'examen à l'université sans faire mention de spécificités liées aux départements ou aux facultés?

2.1.2 Années d'étude ou étudiants?

Comme le problème qui nous occupe est celui de la programmation d'une session d'examen à l'université, il convient de réfléchir à la structure de la base de données à gérer et surtout aux types de relations à prendre en compte. Si on souhaite une solution souple, il faut prendre en compte de nombreuses variables. Une idée qui vient assez naturellement à l'esprit est de considérer que tout étudiant fait partie d'une année d'étude. La modélisation d'une session d'examens où toutes les épreuves sont écrites se limiterait alors à éviter les conflits entre les épreuves des cours d'une même année[23]. Pour des examens oraux, la contrainte est trop sévère. Deux épreuves orales d'une même année peuvent se dérouler en même temps dès lors qu'il n'y a pas de conflits au niveau des individus. Cette simultanéité est d'ailleurs une nécessité dans les facultés qui sont très peuplées.

Voici l'horaire d'**une seule** année d'étude. Cet horaire ne peut convenir puisque deux examens écrits se présentent au cours de la même demi-journée.

Lun am	Écrit C2 !!! Ecrit C3 !!!
Lun pm	
Mar am	Oral C2 (G1)
Mar pm	Oral C2 (G2)
Mer am	Écrit C1
Mer pm	

En revanche, un horaire comme celui qui suit est acceptable à condition qu'aucun étudiant du groupe *G1* pour le cours *C2* ne fasse partie du groupe *G2* pour le cours *C3*.

Lun am	Écrit C2
Lun pm	
Mar am	Oral C2 (G1) Oral C3 (G2)
Mar pm	
Mer am	Oral C1 (G2)
Mer pm	Oral C3 (G1)

[23] Les épreuves des cours *C1* et *C2* dispensés aux étudiants de l'année *A* ne peuvent se dérouler en même temps s'il s'agit d'examens écrits. On suppose que les épreuves écrites concernent la totalité du groupe. On évite ainsi une multiplication inutile des questionnaires tout en assurant une certaine équité entre les étudiants. Si ce n'est pas le cas, la modélisation peut envisager les groupes comme autant de classes différentes.

Il faut remarquer que, dans le cas particulier des cours à option, tous les étudiants ne sont pas nécessairement inscrits à tous les cours et donc à leurs éventuelles épreuves écrites. Dès lors, imposer que deux écrits ne puissent se dérouler au même moment est sans doute trop contraignant.

L'horaire qui suit est acceptable si aucun étudiant inscrit au cours C1 n'est inscrit au cours C3[24].

Lun am	Écrit C2
Lun pm	
Mar am	Oral C2 (G1)
Mar pm	Écrit C1 Écrit C3
Mer am	Oral C2 (G2)
Mer pm	

2.1.3 Groupes anonymes ou individus

Pour tenir compte de ces commentaires, et ainsi éviter des contraintes trop fortes, il est nécessaire d'imaginer que des groupes d'étudiants puissent exister.

Une solution proche de ce qui se passe dans la réalité consiste à faire les considérations suivantes:

- les écrits concernent l'ensemble des étudiants de l'année d'étude,
- pour les oraux, un nombre de groupes à former est à calculer sur base du nombre d'étudiants que le professeur interroge sur un demi-jour,
- un demi-jour est réservé pour chaque groupe à constituer,
- les groupes ne sont pas formés d'avance.

Cette manière de voir les choses pose un certain nombre de problèmes que nous allons examiner.

Les écrits concernent l'ensemble des étudiants de l'année d'étude

C'est évidemment une contrainte beaucoup trop forte. Certains cours sont des cours à option et ne concernent qu'un nombre limité d'étudiants. Dans un horaire conçu sur ces considérations, aucun groupe d'oral ne sera placé en parallèle avec cet examen alors que c'est peut-être possible dans certains cas.

Pour les oraux, un nombre de groupes à former est à calculer sur base du nombre d'étudiants que le professeur interroge sur un demi-jour

Ce calcul n'est pas difficile à réaliser. Il faut toutefois veiller à ce que la répartition dans les groupes se fasse de manière sensiblement égale. Nous décrivons plus en avant un algorithme qui calcule le nombre de groupes à former et avec quel nombre d'étudiants (variable à une unité près).

Un demi-jour est réservé pour chaque groupe à constituer

Dans l'optique où ce sont les étudiants qui se répartissent dans les groupes[25], on admet que des groupes d'oraux se trouvent en parallèle dans l'horaire. Ceci pose un problème dans le cas où un seul groupe est à constituer pour l'oral car les étudiants n'ont pas le choix. Il faut prévoir cette situation et éviter de mettre d'autres épreuves en parallèle avec celles-là.

---

[24] En y regardant de plus près, ce n'est même pas l'inscription au cours qui est déterminante, mais l'inscription aux épreuves de ce cours. Cette remarque est importante dans le contexte des secondes sessions où des reports de notes peuvent avoir lieu.

[25] ...puisque ceux-ci ne sont pas constitués avant le calcul de l'horaire

Les groupes ne sont pas formés d'avance

Ceci correspond assez bien à la réalité de la constitution manuelle des horaires. Le problème est considérablement simplifié mais cela risque de « coincer » pour certains étudiants qui se placent tardivement dans les groupes.

Si de telles considérations sont utiles pour le développement d'une première version, et même si dans la réalité, les choses fonctionnent de cette façon, elles sont inacceptables à terme. Le programme doit fournir des solutions correctes et non des solutions à retoucher. À la limite, les retouches manuelles sont admissibles sur des solutions correctes, dans la mesure où ces retouches améliorent considérablement ces solutions.

L'autre point de vue consiste à abandonner totalement le concept de classe ou d'année d'étude et de ne s'intéresser qu'aux individus. Si on connaît avec exactitude toutes les épreuves auxquelles un étudiant est inscrit, on peut lui fabriquer un horaire individuel. Évidemment, cet horaire individuel devra tenir compte de certaines contraintes telles: le professeur *p* qui interroge en oral pour le cours *c* n'interrogera qu'un nombre de groupes minimum en fonction du nombre d'étudiants qu'il peut interroger par demi-jour.

Exemple: Le professeur dont l'identifiant est *ven* interroge 12 étudiants par demi-jour pour le cours dont l'identifiant est *info2208* et doit interroger 35 étudiants $\Rightarrow$ 4 groupes à former

Une solution simple consiste à répartir préalablement les étudiants dans des groupes. Les contraintes évoquées ci-dessus deviennent alors des contraintes de présence simultanée pour les étudiants d'un même groupe à l'oral.

Exemple: Pour le même cours, on trouvera à l'horaire les éléments suivants:

- oral info2208 G1
- oral info2208 G2
- oral info2208 G3
- oral info2208 G4

Considérer chaque étudiant dans l'expression des contraintes est commode pour les examens oraux. En même temps, cette modélisation règle le problème des étudiants inscrits à des cours dans des années différentes, voire dans des facultés différentes[26]. Pour les écrits, cette modélisation peut paraître lourde car elle impose que tous les étudiants inscrits à une même épreuve soient mobilisés pour cette épreuve au même moment.

Cette multiplication du nombre de variables est audacieuse: une variable pour chaque épreuve de chaque étudiant, cela peut augmenter rapidement en fonction de l'échantillon considéré[27]. La littérature sur le sujet ne tient que trop peu compte d'une programmation individuelle. Pourtant, dans les facultés où les cours à option sont nombreux, cette prise en compte est quasi inévitable. On en trouve un exemple avec confirmation de la chose dans [7]. Cette optique nous amène à réaliser, dans un premier temps, quelques simplifications importantes dans la recherche d'une solution. Nous tâcherons de les justifier. Par la suite, nous montrerons que cette prise en compte des individus n'est pas déraisonnable. Pour le moment, décrivons un modèle qui pourrait être le modèle idéal.

---

[26] Apparemment, il est plus raisonnable d'envisager la programmation d'une session d'examen pour l'ensemble des étudiants et des professeurs de l'université. On peut considérer le problème au niveau d'une faculté ou d'un département si les périodes d'examen ne peuvent être modélisées de la même manière. Dans ce cas, il faut considérer les occupations des professeurs et les obligations des étudiants dans les autres facultés et départements comme des contraintes individuelles fortes.

[27] Au niveau des années de maîtrise à l'Institut d'Informatique des FUNDP, il faut compter environ 180 étudiants ayant une moyenne de 12 à 15 épreuves à présenter. A cela, il faut ajouter les variables concernant les professeurs, ce qui représente de 2 à 3.000 variables.

2.2 Modélisation de la base de données

Considérons de manière simple que les ressources du problème sont de trois sortes:

- les **locaux** (dont l'importance est toute relative dans le contexte qui nous occupe),
- les **personnes** (professeurs comme étudiants),
- les **périodes** disponibles (la session n'étant pas d'une durée illimitée).

En tout généralité, le simple fait de fixer la date d'une épreuve d'examen monopolise donc un certain nombre de ces ressources: un (ou plusieurs) examinateur(s)[28], des étudiants, une (ou plusieurs) période(s)[29], un (ou plusieurs) local (locaux).

Examinons ces différentes ressources.

2.2.1 Les examinateurs

Les examinateurs font passer des épreuves correspondant à des cours.

Par facilité, nous assimilerons les concepts de *professeur* et d'*examinateur*, ce qui nous permettra de faire plus facilement l'association avec les cours[30]. Les professeurs peuvent être identifiés sans trop de difficultés. Pour les cours, nous considérerons que la situation est intermédiaire entre des cours associés à une année d'étude et des cours modulaires. Bien que ces cours soient annoncés dans le programme pour les étudiants d'une année bien particulière, nous admettrons qu'ils puissent être suivis par des étudiants d'une autre année[31] et que ceux-ci puissent en passer les épreuves. De ce fait, si la dénomination d'un cours comprend une référence à l'année officielle, c'est pour désigner le cours sans équivoque. Ainsi, si *info2210* désigne un cours de deuxième maîtrise, cela ne veut pas nécessairement dire que des étudiants de première ne pourraient pas le suivre et en passer l'épreuve. À terme, les dénominations des cours peuvent ne plus comporter de référence à l'année et faire place à des noms de modules sans que la modélisation soit à revoir.

Les attributions des enseignants se traduiront par des faits du genre:

enseigne(Nom_prof, Liste_de_cours)

comme par exemple,

enseigne(jpl, [info2101, info2201, info2220, info2222])

Le professeur dont l'identifiant est *jpl* enseigne les cours dont les identifiants sont *info2101*, *info2201*, *info2220* et *info2222*.

Ces éléments sont les premiers qui caractérisent la base de données concernant les examens. La structure de cette base de données sera résumée plus loin dans ce chapitre.

Les examinateurs sont indisponibles certains jours ou demi-jours de la session.

Les indisponibilités des professeurs seront traduites comme suit:

indispo(Nom_professeur, Liste_de_demi-jours)[32]

---

[28] Le terme "examinateur" convient sans doute mieux, en toute généralité, que le terme "professeur".

[29] En théorie, une seule période est monopolisée mais on peut imaginer qu'un examen soit contraignant au point d'empêcher un autre examen de se trouver trop proche de lui.

[30] Une modélisation plus sophistiquée pourrait prendre en compte les personnes susceptibles de faire passer l'épreuve ainsi que leurs disponibilités.

[31] Il s'agit ici d'une possibilité qui, dans l'état actuel des choses, est soumise à autorisation. Il ne s'agit pas d'un droit des étudiants à présenter les modules qu'ils souhaitent. Toutefois, cela ne change rien du point de vue de la modélisation.

[32] La notion de liste trouve tout son intérêt avec les langages de programmation logique.

comme par exemple

indispo(jmj,[1,2,3,4,13,14])

Le professeur dont l'identifiant est *jmj* ne peut interroger les quatre premiers demi-jours de la session ni les treizième et quatorzième demi-jours.

2.2.2 Les épreuves

Le professeur fait passer les épreuves correspondant à ces cours. Les cours seront caractérisés par le type d'épreuves à présenter.

Les types d'épreuves seront traduits de la manière suivante:

epreuve(Cours, Type)

comme par exemple,

epreuve(info2105,ecrit+oral(8))

Les épreuves du cours dont l'identifiant est *info2105* sont:

- un écrit
- un oral pour lequel le professeur accepte 8 étudiants par demi-jour

ou encore,

epreuves(info2110,oral(12))

Une seule épreuve liée au cours *info2110*: un oral pour lequel le professeur interroge 12 étudiants par demi-jour.

Pour couvrir un ensemble significatif de cas[33], nous envisageons les possibilités suivantes:

- un écrit seul (traduit par *ecrit*)
- un oral seul (traduit par *oral(N)* où *N* est le nombre d'étudiants interrogés)
- un écrit et un oral qui le suit dans la session (traduit par *ecrit+oral(N)*)
- un ecrit et un oral unique de rattrapage le même jour (traduit par *ecrit*oral*)

Ajoutons encore que si un professeur ne souhaite qu'un seul demi-jour d'oral quel que soit le nombre d'étudiants, il lui suffit d'indiquer un nombre d'étudiants suffisamment élevé à interroger par demi-jour (conventionnellement, 999).

Cette manière de décrire les épreuves s'avère assez riche dans la mesure où elle permet une gestion implicite des cas particuliers. Si un professeur souhaite que l'écrit d'un cours soit précédé par l'oral, on peut décrire les épreuves de ce cours d'une autre manière, de façon à permettre au programme de ne pas tenir compte de certaines contraintes pour certains cours. Par exemple:

epreuve(info2210, oral(9)+ecrit)

La formulation *oral(N)+ecrit* n'ayant pas encore été utilisée, elle pourrait autoriser des cas particuliers.

2.2.3 Les étudiants

Les étudiants sont inscrits à des cours à l'issue desquels ils passent une ou des épreuves (orale, écrite ou les deux).

Dans un modèle très simple, nous pourrions considérer que les étudiants font partie d'une année d'étude (par exemple, la *1ère maîtrise en informatique*) et que la programmation d'un horaire doit tenir

---

[33] La modélisation pourrait s'enrichir dans la mesure où d'autres contraintes, au niveau des épreuves, pourraient être traduites par d'autres expressions à analyser.

compte du fait que deux épreuves de la même année ne peuvent se dérouler en même temps. Nous l'avons déjà signalé, ce modèle est trop simple et ne permet pas, par exemple, que deux examens oraux des étudiants d'une année très peuplée se déroulent simultanément.

Dans un modèle général, nous considérerons que chaque étudiant est inscrit à un ensemble de cours pour lesquels il doit présenter un certain nombre d'épreuves. Ces cours peuvent être répertoriés dans le programme de deux, voire plusieurs années[34].

Le fait qu'un étudiant soit inscrit à des cours se traduit de la manière suivante:

inscrit(Nom_étudiant, Liste de cours)

comme par exemple,

inscrit(evandeput,[info2201, info2202, info2212]).

qui indique que l'étudiant dont l'identifiant est *evandeput* suit les cours dont les identifiants respectifs sont *info2201*, *info2202* et *info2212*.

2.3 Modélisation des relations

2.3.1 Entre les étudiants et les cours

Chaque triplet *(étudiant, cours, type d'épreuve)* doit se voir attribuer une valeur unique comme moment d'examen. Cette valeur unique sera précisément la valeur finale attribuée à une variable de domaine fini. Les données qui précèdent doivent donc permettre de déduire des faits ayant la structure suivante:

passe(Nom_etudiant, Cours, Type, Variable_FD)

Ainsi, par exemple, le fait suivant

passe(evandeput, info2105, ecrit, X)

établit une liaison entre l'épreuve écrite que doit passer l'étudiant dont l'identifiant est *evandeput* pour le cours dont l'identifiant est *info2105* et une variable X de domaine fini dont la valeur devra être déterminée[35].

Cette modélisation permet de prendre en compte de manière simple les cas où les étudiants suivent de nombreux cours à option.

2.3.2 Entre les professeurs et les cours

Chaque triplet *(professeur, cours, type d'épreuve)* doit se voir attribuer une valeur unique comme moment d'examen. Cette valeur unique sera précisément la valeur finale attribuée à une variable de domaine fini. Les données qui précèdent doivent donc permettre de déduire des faits ayant la structure suivante:

examine(Nom_professeur, Cours, Type, Variable_FD)

Dans cette structure, il est bon de s'intéresser à la signification du *Type*. Alors que pour un étudiant, ce type ne peut prendre que les deux valeurs *ecrit* ou *oral*, il n'en est pas de même pour les professeurs. Si nous voulons associer une variable FD à un triplet *(professeur, cours, type d'épreuve)*, nous devons obligatoirement distinguer toutes les interventions du professeur dans le cours en question. Ainsi,

---

[34] Cette considération pose tout de même un petit problème de cohérence de données. Il existe en effet une différence entre les cours auxquels les étudiants sont inscrits et ceux pour lesquels ils doivent présenter les examens. C'est le cas des étudiants répertoriés dans au moins deux années d'études et des étudiants qui bénéficient de reports de notes.

[35] A partir du triplet de données, la variable peut être retrouvée sans peine.

dans le cas d'un oral, celui-ci doit souvent consacrer plusieurs demi-jours d'interrogation. De ce fait, *Type* peut prendre la valeur *ecrit* et un certain nombre de valeurs dépendant du nombre de groupes à interroger: *oral(1)*, *oral(2)*,..., *oral(n)*.

À titre d'exemple,

examine(jpl, info2201, ecrit, X)

établit une liaison entre l'épreuve écrite que doit faire passer le professeur dont l'identifiant est *jpl* pour le cours dont l'identifiant est *info2201* et une variable *X* de domaine fini dont la valeur devra être déterminée.

De même,

examine(ven, info2208, oral(2), X)

établit une liaison entre l'épreuve orale du deuxième groupe constitué que doit faire passer le professeur dont l'identifiant est *ven* pour le cours dont l'identifiant est *info2208* et une variable *X* de domaine fini dont la valeur devra être déterminée.

2.4 Modélisation des contraintes

2.4.1 Les contraintes indirectes

Rappelons qu'il s'agit de contraintes qui ne sont pas spécifiques au domaine d'application mais qui relèvent de la disponibilité des ressources. Rappelons aussi que nous considérons les personnes (professeurs et étudiants) comme des ressources qui ne peuvent être sollicitées au même moment pour deux activités différentes.

Nous admettrons que **les professeurs font passer les épreuves des cours qu'ils dispensent.**

Nous considérerons que le professeur qui enseigne le cours *Cours* est aussi la personne qui en fait passer les épreuves. De ce fait, les moments accordés au professeur *Nom_du_professeur* pour faire passer l'épreuve de *Cours* ne peuvent être associés à un autre triplet *(Nom_du_professeur, Cours, Type)*. Si le professeur *Nom_du_professeur* peut se faire remplacer, la contrainte disparaît mais il faut alors régler le problème de la disponibilité du remplaçant. Une modélisation fine pourrait envisager d'associer une liste d'interrogateurs potentiels à certains cours.

C1. Un examinateur ne peut faire passer deux épreuves différentes au même moment.

Si cette affirmation est incontestable, sa traduction dans le contexte est légèrement différente. Si on s'intéresse à l'affirmation: « Un professeur ne peut interroger en même temps deux groupes correspondant aux cours qu'il enseigne », elle est plus contestable car:

- pour des écrits, il pourrait interroger plusieurs groupes en même temps,
- pour tous les types d'examens, il pourrait se faire suppléer par un assistant.

L'affirmation est donc vraie si le nom du chargé de cours est confondu avec celui de l'examinateur sauf si celui-ci décide de regrouper plusieurs examens écrits au même moment et au même endroit[36]. Si ce n'est pas le cas, il faut associer une liste d'examinateurs potentiels à chaque cours et gérer les disponibilités de chacun d'entre eux. On peut admettre, par exemple, que pour les examens écrits le titulaire du cours puisse faire appel à des assistants. La contrainte est ainsi moins forte. On peut aussi, dans les données de base, associer un autre nom que celui de l'examinateur (ou même d'autres noms dérivés: *mno1*, *mno2*...) aux épreuves écrites pour lesquelles le professeur souhaite déléguer la

---

[36] Cette contrainte est délicate car elle suppose que les écrits concernés se rapportent fort probablement à des années d'études différentes et peut donc entrer en conflit insoluble avec le fait qu'un étudiant puisse passer des épreuves correspondant à d'autres années d'études que celle dans laquelle il est inscrit en principal.

surveillance. Ce dernier déciderait alors, en cas de conflit d'horaires entre les épreuves qu'il organise, à quelle épreuve il est présent et à quelles autres il se fait représenter.

Exemple: si le professeur dont l'identifiant est *mno* n'exige pas d'être présent à l'épreuve écrite du cours *info2105* mais bien à l'épreuve écrite de *info2202*, les données peuvent être enregistrées sous la forme

enseigne(mno,info2202)

enseigne(mno1,info2105)

Le professeur choisit alors de se faire remplacer pour le second examen en cas de nécessité.

Très concrètement, si on impose que le professeur soit présent à tous ses examens et qu'il n'interroge pas deux groupes simultanément, cette contrainte se traduit dans la formulation suivante:

$$examine(Id\_Prof, *, *, X) \wedge examine(Id\_Prof, *, *, Y) \Rightarrow X \neq Y$$

Dans cette expression comme dans celles qui suivent, le symbole * est générique et X et Y désignent des variables FD différentes.

Comme vous pouvez le constater, dans l'approche choisie, les contraintes agissent au niveau des variables de domaine fini.

C2. Un professeur ne peut interroger un demi-jour où il est indisponible.

Les demi-jours d'indisponibilité d'un professeur sont énumérés dans une liste. Il faudra donc que

$$examine(Id\_Prof, *, *, X) \wedge indispo(Id\_Prof, L) \wedge n \in L \Rightarrow X \neq n$$

À nouveau, la contrainte agit sur le domaine de certaines variables FD.

C3. Un étudiant ne peut passer deux épreuves différentes au même moment.

Contrairement à la précédente, cette contrainte est vraiment incontestable. Dès lors, on doit admettre que:

$$passe(Id\_Etudiant, *, *, X) \wedge passe(Id\_Etudiant, *, *, Y) \Rightarrow X \neq Y$$

C4. Les épreuves se déroulent le matin et/ou l'après-midi de chacun des jours de la session d'examens.

Le nombre de jours d'une session d'examens est fixé à N (longueur maximum supposée, par exemple 40 jours). Nous définirons donc les moments possibles d'examen comme les matinées ou les après-midi de ces jours. Ils peuvent être numérotés de 1 à $2N$. Une découpe plus fine est envisageable en augmentant le coefficient de N. La découpe en demi-jours est intéressante si on demande aux enseignants combien d'étudiants ils peuvent interroger le matin et l'après-midi lors de leurs examens oraux.

$$examine(*, *, *, X) \Rightarrow X \in [1,2N]$$

$$passe(*, *, *, X) \Rightarrow X \in [1,2N]$$

C5. Le professeur et les étudiants concernés par un même examen le sont au même moment.

C'est sans doute la contrainte la plus évidente mais c'est celle qui relie les exigences émises au niveau des professeurs à celles émises au niveau des étudiants et qui permet, dans le même temps, de considérer chaque étudiant individuellement.

$$examine(*, Cours, T, X) \wedge passe(*, Cours, T, Y) \Rightarrow X = Y$$

Pour rappel, T peut prendre les valeurs *ecrit*, *oral(1)*, *oral(2)*,..., *oral(n)* si l'épreuve orale demande la formation de n groupes.

Il faut traduire: « n'importe quel étudiant suivant inscrit au cours *Cours* et passant l'épreuve *T* le fait le demi-jour pendant lequel le professeur du cours *Cours* interroge pour cette épreuve (et éventuellement ce groupe) ».

2.4.2 Les contraintes pédagogiques

Il s'agit de contraintes dont il faudra juger, dans une implantation raisonnable, si elles sont vraiment requises. Nous en citons quelques-unes pour exemple, l'essentiel étant de montrer que ces contraintes se traduisent assez simplement, et comme les contraintes indirectes, au niveau des variables FD. Cela signifie qu'un développement de type incrémental est permis.

CP1. Lorsqu'à un cours, sont associés un examen écrit et un examen oral, l'écrit précède toujours l'oral.

Cette contrainte est justifiée par le fait que l'examen oral peut permettre de vérifier des états de connaissance et de compréhension déduits de l'épreuve écrite.

Si cette contrainte devait être évitée pour certains cours, on pourrait choisir une modélisation plus fine des types d'épreuves qui évite la prise en compte de cette contrainte pour les cours de ce type nouveau.

$$examine(*, Cours, ecrit, X) \wedge examine(*, Cours, oral(G), Y) \Rightarrow X < Y$$

Dans cette proposition, *G* désigne un numéro de groupe.

On peut imaginer qu'un certain délai soit nécessaire entre les deux épreuves. Ce problème peut également être résolu par une modélisation plus fine des types d'épreuves.

CP2. Un professeur ne peut faire passer deux épreuves différentes le même jour.

Le professeur *Id_Prof* ne fera pas passer un examen de *Cours1* le matin et de *Cours2* l'après-midi.

Sachant que les matinées correspondent à des nombres impairs et les après-midi à des nombres pairs. Si *X* et *Y* représentent des variables de domaine fini correspondant aux périodes de deux épreuves différentes, les contraintes seront

$$examine(Nom, Cours1, *, X) \wedge examine(Nom, Cours2, *, Y) \wedge X \bmod 2 = 0 \Rightarrow Y \neq X-1$$

$$examine(Nom, Cours1, *, X) \wedge examine(Nom, Cours2, *, Y) \wedge X \bmod 2 = 1 \Rightarrow Y \neq X+1$$

CP3. Un étudiant ne doit pas passer deux épreuves d'examen le même jour.

Cette contrainte peut évidemment être renforcée ou affaiblie en cas de nécessité. « L'écart entre deux examens consécutifs d'un étudiant est d'au moins *n* demi-jours ». Sans que cette contrainte soit considérée comme une contrainte douce, ses paramètres peuvent être revus à la hausse ou à la baisse (valeur minimum de *n*: 1) selon que le problème admet ou n'admet pas de solution.

$$passe(Id\_Etudiant, *, *, X) \wedge passe(Id\_Etudiant, *, *, Y) \Rightarrow dist(X,Y) >= n$$

On peut s'interroger sur le bon sens d'une telle contrainte. L'importance et la difficulté relatives des examens font que cette contrainte est parfois inutile, parfois trop peu sévère. Nous proposons une voie qui nous paraît intéressante dans le chapitre 6 consacré aux perspectives.

Pour assurer une bonne répartition des examens dans la session d'un professeur et d'un étudiant, une série de contraintes supplémentaires pourraient être prises en compte. Remarquez que la cohérence d'un horaire pour le professeur et la cohérence d'un horaire pour l'étudiant ne signifient pas la même chose. L'étudiant souhaitera une bonne répartition des examens sur l'ensemble des jours de la session alors que le professeur souhaitera un certain regroupement pour les oraux d'un même cours.

Citons en vrac quelques-unes des contraintes de cette nature qui pourraient être prises en compte sans qu'elles soient nécessairement admises par les principaux intéressés.

CP4. La répartition des examens d'un étudiant doit s'étendre sur au moins N% du temps de la session.

Cette contrainte est une contrainte paramétrée (par la valeur de *N*).

Par exemple, si le nombre total de périodes est 80 et que *N* est fixé à 75, la session de l'étudiant doit s'étendre sur au moins 60 périodes.

On devine que si une telle contrainte est requise, on peut rater une solution de fort peu. Imaginons qu'une solution soit possible en fermant les yeux sur le fait que pour un étudiant, ce pourcentage ne soit pas tout à fait atteint, on aimerait qu'une telle contrainte soit adoucie et que son poids, par exemple, soit proportionnel au dépassement du pourcentage. Si les contraintes douces ne sont pas acceptées, il vaut mieux qu'elle soit évitée et que la tâche de bonne répartition soit confiée au solveur de contraintes par le biais d'une heuristique appropriée[37].

CP5. Si l'étudiant présente au moins cinq examens, l'écart entre deux examens ne peut excéder le double du nombre total de périodes divisé par le nombre d'épreuves à passer.

Cette contrainte fait partie des contraintes qui peuvent être paramétrées différemment, voire négligées en cas de nécessité.

Si le nombre total de périodes est 80 et que l'étudiant a six épreuves à présenter, l'écart entre deux épreuves ne peut excéder 26 périodes.

On le voit, la plupart de ces contraintes sont basées sur l'intuition et la manière la plus objective de les définir semble être de donner une limite extrême à ne pas franchir tout en leur attribuant un poids proportionnel à ce dépassement.

CP6. Si le professeur fait passer des épreuves pour plusieurs cours, les périodes de celles-ci ne doivent pas s'entrecroiser.

Si le professeur *Id_Prof* fait passer des examens pour les cours *Cours1* et *Cours2*, les moments attribués à l'examen de *Cours1* ne doivent pas se mélanger aux moments attribués à l'examen de *Cours2*.

Une séquence comme la suivante n'est pas admise: *Cours2 Cours2 * Cours1 Cours2 * * Cours2 Cours1 Cours1 Cours2*

À nouveau, on constate qu'une telle contrainte de pseudo-contiguïté se justifie dans certains cas et pas dans d'autres. On pourrait éventuellement en faire une contrainte « à option ».

L'utilisation des variables FD facilite l'expression d'une telle contrainte.

CP7. Les moments consacrés aux examens oraux doivent être, autant que possible, consécutifs.

Dans un modèle général, on se donnera la possibilité de fixer un nombre de périodes à ne pas dépasser entre deux moments d'interrogation consécutifs d'un même cours. On veillera aussi à ce que les séances prévues pour un même examen ne s'étendent pas sur une trop longue période de temps.

On peut exiger qu'il ne se trouve pas plus de *n* périodes entre deux séances d'interrogation successives concernant le cours *Cours*.

Si l'épreuve orale du cours *Cours* demande *m* périodes, le longueur totale L de la période d'interrogation pour le cours *Cours* ne doit pas dépasser *rnd(αm)* périodes.

Par exemple, si le cours requiert la constitution de 3 groupes pour l'oral ($\alpha = 1,8$), ces trois groupes devront être interrogés sur un intervalle de 5 demi-jours.

$$m=3 \wedge \alpha=1,8 \Rightarrow L=rnd(1,8*3) = 5$$

---

[37] Une heuristique basée sur un choix aléatoire des valeurs pour les variables FD donne d'assez bons résultats du point de vue de la répartition des épreuves pour les étudiants.

Cette contrainte peut être optionnelle si on admet que certains interrogateurs ne souhaitent pas concentrer leurs périodes d'examens.

Ces dernières conditions peuvent apparaître comme des contraintes douces si celles-ci sont autorisées.

2.4.3 Contraintes indirectes et pédagogiques concernant les locaux

Nous avons évité jusque là les contraintes concernant les locaux mais dans un contexte très général, la constitution d'un horaire ne va pas sans une répartition de ceux-ci. Dans le cas de l'horaire d'une session d'examen, ce problème peut être oublié car la concurrence est faible, voire inexistante. Nous citons toutefois quelques contraintes qui devraient être prises en compte dans un contexte plus général.

Un local ne peut accueillir deux examens pendant les mêmes périodes de temps.

Cette affirmation ne tient pas si on admet que plusieurs examens écrits d'un même professeur puissent se dérouler en même temps, au même endroit. Nous avons déjà évoqué les conflits que cette contrainte pouvait faire surgir.

Certaines épreuves doivent absolument se dérouler dans un local précis ou un local vérifiant certaines conditions

Un local précis peut être requis pour un examen. Le choix d'un local peut aussi être lié à certaines contraintes telles le nombre d'étudiants pour une épreuve écrite unique, la présence d'un ou de plusieurs tableaux pour un oral,... La structure des faits traduisant les caractéristiques des épreuves d'un cours peut être améliorée. Par exemple:

epreuve(info2110,ecrit,[i1,i2])

epreuve(info2110,oral(10),[])

L'épreuve écrite du cours *info2110* doit absolument se dérouler soit dans le local *I1* ou dans le local *I2* et l'épreuve orale peut avoir lieu dans n'importe quel local (liste vide).

Pour les examens écrits, la population du cours est généralement un facteur important. Les locaux peuvent être décrits avec un certain nombre de caractéristiques comme, par exemple:

local(i2,50,tableau(3),[retro,pc])

peut signifier que le local *I2* compte 50 places, 3 tableaux et comme matériel supplémentaire, un rétroprojecteur et un ordinateur de type *PC*.

À nouveau, on peut constater qu'au plus le schéma de la base de données se traduit finement au niveau de la fourniture des données, au plus le programme peut en tenir compte par l'intégration de contraintes simples concernant les variables FD du problème.

2.4.4 Création de groupes

La programmation d'une épreuve orale nécessite la création de groupes qui se verront attribuer une date d'examen différente pour la même épreuve. Des contraintes apparaissent pour le contrôle du nombre d'individus par périodes. Ce nombre ne peut être supérieur au nombre d'étudiants que le professeur compte interroger par demi-journées. Il est même souhaitable que les étudiants soient répartis de manière plus ou moins égale dans les différentes périodes pour éviter le déplacement d'un professeur pour un ou deux étudiants.

Exemple:

Il y a 18 étudiants à interroger à raison de 8 étudiants par demi-jour.

La répartition 8 8 2 peu souhaitable!

Si le professeur *P* interroge *N* étudiants par demi-journées, les étudiants seront répartis en groupes de tailles égales (à une unité près) et les plus proches possible de *N*.

Voici quelques exemples de répartitions pour N=10. Le premier nombre représente le nombre total d'étudiants à interroger:

```
40: 10 10 10 10
39: 10 10 10 9
38: 10 10 9 9
...
32: 8 8 8 8
31: 8 8 8 7
30: 10 10 10
...
12: 6 6
11: 6 5
10: 10
...
```

Le calcul du nombre d'étudiants à caser dans une période peut s'effectuer en utilisant le petit algorithme suivant:

Soit *NTot* le nombre d'étudiants inscrits au cours concerné et *N* le nombre d'étudiants que le professeur accepte d'interroger sur un demi-jour.

```
N1 = Ntot div N
R1 = Ntot mod N
Si R1 = 0 alors former N1 groupes de N étudiants
Sinon
      N2 = NTot  div (N1 + 1)
      R2 = NTot mod (N1 + 1)
      Si R2 = 0 alors former (N1 + 1) groupes de N2 étudiants
      Sinon
            former (N1 - R2 + 1) groupes de N2 étudiants
            former R2 groupes de (N2 + 1) étudiants
```

Exemples (20 étudiants interrogés par demi-jour)		
113 étudiants inscrits	100 étudiants inscrits	108 étudiants inscrits
NTot = 113 et *N* = 20	*NTot* = 100 et *N* = 20	*NTot* = 108 et *N* = 20
N1 = *NTot* div 20 = 5	*N1* = *NTot* div 20 = 5	*N1* = *NTot* div 20 = 5
R1 = *NTot* mod 20 = 13	*R1* = *NTot* mod 20 = 0	*R1* = *NTot* mod 20 = 8
N2 = *NTot* div 6 = 18	Solution: 5 groupes de 20	*N2* = *NTot* div 6 = 18
R2 = *NTot* div 6 = 5		*R2* = *NTot* div 6 = 0
Solution: 1 groupe de 18 et 5 groupes de 19		Solution: 6 groupes de 18

2.5 Les grandes simplifications du modèle

2.5.1 Pas de contraintes douces

Nous ne tiendrons pas compte d'éventuelles contraintes douces ou d'une quelconque hiérarchie de contraintes. Elles existent pourtant: professeur souhaitant interroger tel jour plutôt que tel autre, épreuves d'examens idéalement séparées de **x** jours,... La réalité de la constitution manuelle des horaires voudrait laisser croire que les contraintes de l'ordre du souhait sont généralement ignorées, ce qui nous donne bonne conscience dans cette première approximation.

2.5.2 Omission des contraintes d'attribution des locaux

Les contraintes liées à l'attribution des locaux sont également omises. Dans la recherche d'une solution de programmation d'horaire comme dans [3], le problème de l'attribution des locaux peut être complètement dissocié du problème d'horaire proprement dit, soit que la méthode de résolution est basée sur l'amélioration d'une solution antérieure (auquel cas un conflit de locaux est vu comme une amélioration), soit que l'on considère que le problème de l'attribution des locaux est aisément résolu, même manuellement, vu la faible proportionnalité des épreuves écrites et la grande disponibilité de petits locaux pour les besoins des épreuves orales.

2.5.3 Pas d'écrits simultanés pour un même professeur

Les raisons essentielles sont que certains étudiants pourraient être concernés par plusieurs de ces examens. Évidemment, on peut toujours rendre le professeur responsable de sa demande en ce sens qu'il devrait lui-même vérifier qu'aucun étudiant ne se trouve dans deux groupes. De plus, l'état actuel de la demande est faible et se rapporte souvent à des examens concernant les étudiants de facultés différentes.

2.6 La structure des données

Les données nécessaires à l'établissement de l'horaire peuvent se ramener à plusieurs listes:

- une liste décrivant les épreuves associées aux cours qui aura la forme

[epreuve(info2109,oral(9)), epreuve(info2211,ecrit+oral(10)),...]

- une liste décrivant les cours enseignés par les professeurs qui aura la forme

[enseigne(jpl, [info2101, info2201, info2220, info2222]),enseigne(clo,[info2102,info2304]),...]

- une liste décrivant les étudiants inscrits au cours et plus précisément aux examens qui aura la forme

[inscrit(achbany,[info2229,ielv2212,...]),inscrit(anciaux,[ielv2212,...]),...]

Un étudiant peut n'être inscrit à aucun cours. De même, un cours peut ne compter aucun étudiant. Un cours peut ne pas être attribué. Un professeur peut ne pas avoir de cours. Un cours comporte zéro, une ou deux épreuves. Une épreuve, surtout si elle est orale peut avoir plusieurs instances (groupes d'étudiants. Cette instance se produit lors d'un seul demi-jour. Les professeurs sont indisponibles un certain nombre de jours.

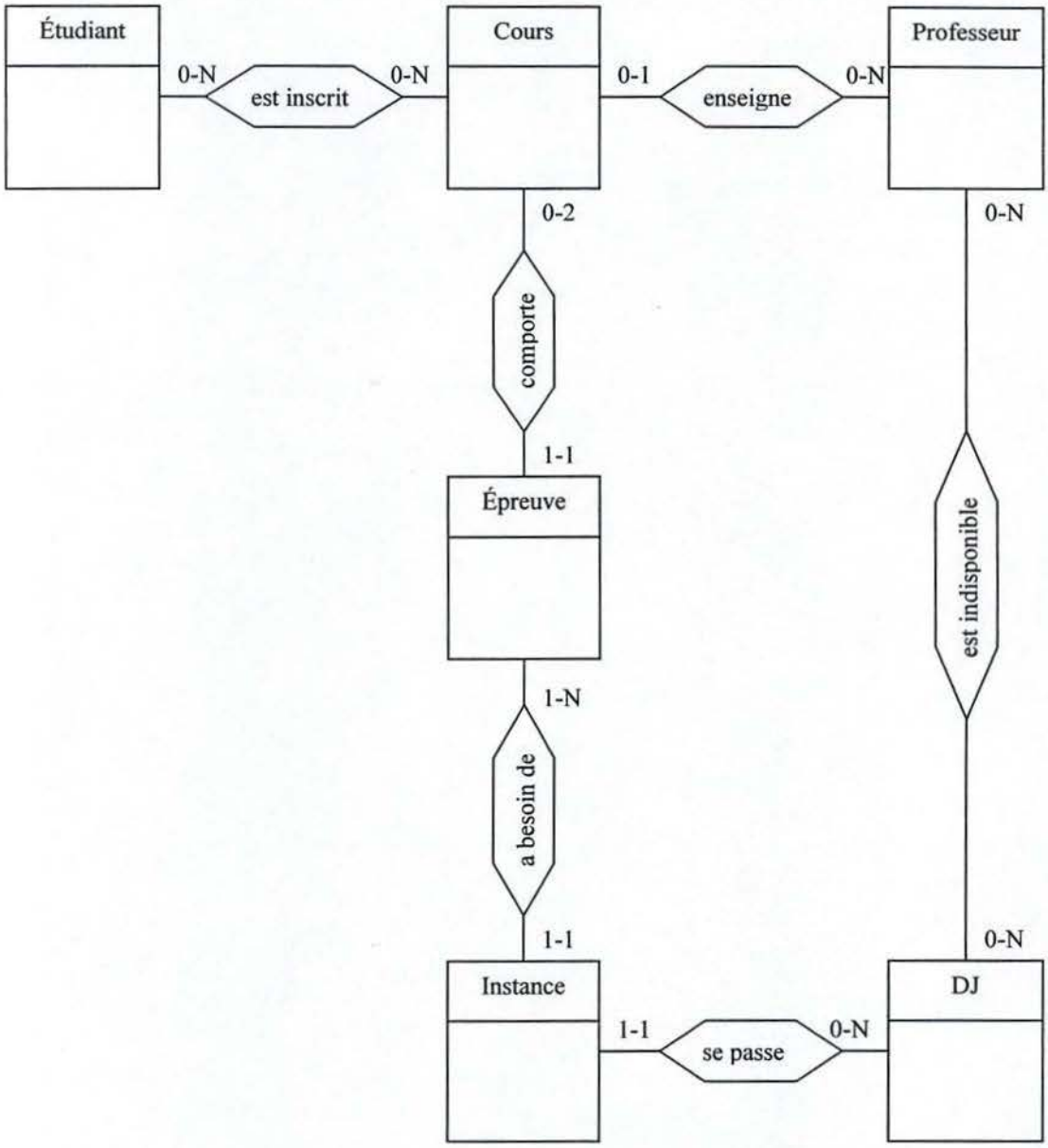
Une quatrième liste est nécessaire, celle qui décrit les indisponibilités des professeurs. Elle est du type:

[indispo(jmj,[1,2,3,4,13,14]), indispo(jpl,[...]),...]

Une liste plus utilitaire, permet de convertir les numéros des demi-jours en dates. Elle ressemble à

[date(1, '31 mai am'), date(2, '31 mai pm'), date(3, '1 juin am'),...]

Voici un schéma de la base de données mettant en évidence les relations qui viennent d'être mentionnées.



Chapitre 3 IMPLANTATION EN PROLOG

C'est parce que l'on imagine tous les pas qu'on devra faire qu'on se décourage, alors qu'il s'agit de les aligner un à un.
(Marcel Jouhandeau)

3.1 Implantation

3.1.1 GNU-Prolog

La nature même du problème fait qu'il relève à la fois d'un certain nombre de constats (attributions, inscriptions, contraintes...) et de la nécessité d'effectuer des calculs (au sens large). Le choix d'un langage qui allie la souplesse de la programmation déclarative avec une certaine puissance de calcul semble s'imposer.

L'implantation est réalisée en *GNU-Prolog*, un compilateur *Prolog* du domaine libre qui inclut un solveur de contraintes sur les domaines finis.

Le problème peut s'exprimer simplement de la manière suivante: chaque épreuve de chaque étudiant doit se voir attribuer une période parmi les $2N$ périodes disponibles[38]. Chaque épreuve que chaque étudiant doit présenter est donc assimilée à une variable qui doit prendre une valeur entre 1 et $2N$.

Il en est de même des épreuves que font passer les enseignants, puisque les épreuves écrites se déroulent sur une seule période et les épreuves orales sur plusieurs périodes. Chaque épreuve que doit faire passer chaque enseignant[39] doit également se voir attribuer une valeur dans le même intervalle de départ.

Toutes les contraintes sont exprimées en fonction de ces variables. Une difficulté est d'indiquer au programme dans quel ordre il doit se préoccuper des variables et comment il doit choisir les valeurs au moment où ce choix doit être réalisé.

3.1.2 Qualités du langage GNU-Prolog

GNU Prolog accepte les prédicats classiques de *Prolog* et propose une série de prédicats liés à la gestion des contraintes. Il permet de produire du code binaire sous forme de programmes exécutables autonomes. La taille d'un exécutable peut être considérablement réduite par le fait que les prédicats non utilisés par le programme ne sont pas « linkés ». Un interpréteur interactif (*top-level*) est également disponible et s'avère utile lors de la mise au point des programmes. Les prédicats purement *Prolog* sont conformes au standard *ISO* pour *Prolog*. Des prédicats supplémentaires sont également disponibles, tels: une interface avec le système d'exploitation, la possibilité d'utiliser des variables globales,...

L'autre partie de *GNU Prolog* comprend un solveur de contraintes sur les domaines finis qui associe la souplesse du déclaratif à la rigueur du calcul dans une programmation logique par contraintes.

---

[38] Pour rappel, N représente le nombre de jours de la session.

[39] Pour un examen oral, il y a autant d'épreuves que de groupes à interroger.

3.2 Structure générale du programme

3.2.1 Description

Le programme doit s'atteler à trouver une valeur acceptable pour chacune des variables de domaine fini correspondant aux différentes épreuves de chaque étudiant et aux différentes prestations (une par examen écrit et souvent plusieurs par examen oral) de chaque enseignant.

Ces variables doivent être accessibles à tout moment, qu'il s'agisse de leur appliquer des contraintes ou d'en fixer les valeurs.

Le module principal du programme décrit les quatre modules principaux mis en œuvre séquentiellement lors de son exécution:

- **construction de listes** FDE **et** FDP d'éléments contenant des références à **des variables de domaine fini** correspondant aux demi-jours d'examen pour les étudiants et les professeurs
- **description des contraintes** à appliquer à ces variables[40]
- **labeling** des variables
- **impression** des résultats dans des fichiers

3.2.2 Code et commentaires

Le prédicat *horaire* réussit si une instanciation de toutes les variables (au niveau de la procédure *label*) est possible.

```
horaire:-
    write('\n\nCréation des listes de variables FD...\n\n'),
    listesFD(FDE,FDP),
    write('Prise en compte des contraintes...\n\n'),
    contraintes(FDE,FDP),
    write('Début de la labelisation...\n\n'),
    label(FDE,FDP),
    write('Impressions dans un fichier...\n\n'),
    impressions(FDE,FDP),
    write('TERMINE').
```

Le prédicat *label* comporte en réalité deux versions, une version pour la constitution de groupes anonymes (pas de labélisation des variables associées aux étudiants) et une version pour la constitution d'horaires individuels pour les étudiants. Ces deux versions sont détaillées dans le paragraphe consacré à la labélisation.

Ce code appelle peu de commentaires si ce n'est de préciser que les listes *FDE* et *FDP* sont les listes qui associent les variables FD aux données les concernant.

FDE: liste associative contenant des références aux variables FD des étudiants

FDP: liste associative contenant des références aux variables FD des professeurs

Une fois créées grâce au prédicat *listesFD*, ces listes sont réutilisées pour la fixation des contraintes, puis des valeurs et enfin pour l'impression. La séquence de caractères *\n* provoque un retour à la ligne.

---

[40] Par facilité, certaines contraintes élémentaires sont déjà fixées dès la création des variables FD (pas d'examen le dimanche, par exemple).

3.3 Création des variables FD

3.3.1 Description

Le module crée les variables FD à partir des données contenues dans les fichiers *profs.txt*, *epreuves.txt* et *inscr.txt*.

Pour chaque triplet (*Nom_étudiant*, *Cours*, *Type*), deux variables sont créées:

- la valeur du demi-jour d'examen
- le numéro du groupe auquel il appartiendra si c'est une épreuve orale en plusieurs groupes (ce numéro sera inclus dans l'information sur le type)

Exemple:

passe(evandepu, info2233, oral(G), X)

La variable *X* correspond à la date de passage de l'étudiant et la variable *G* au numéro du groupe auquel il appartiendra.

Pour chaque triplet *(Nom_professeur, Cours, Type)*, une variable est créée:

- la valeur du demi-jour d'examen

Exemple:

examine(fbo, info2233, oral(1), X1)

examine(fbo, info2233, oral(2), X2)

...

Il est à remarquer que, dans ce cas, la valeur de *G* est fixée puisque le nombre de groupes à former peut être connu au moment de la génération des variables FD.

Exemple:

Si, pour un cours donné, il y a une épreuve écrite et 5 groupes à former pour l'oral, il y aura 6 variables FD à créer et à associer au professeur de ce cours.

L'association des variables au sein des listes *FDE* et *FDP* se fait de la manière suivante:

- des listes *Prolog* sont créées sur base du contenu des fichiers textes
    - *LP*: liste des professeurs

    la liste *LP* contient des éléments du type:

    enseigne(Nom_professeur, Liste_de_cours)

    - *LE*: liste des étudiants

    la liste *LE* contient des éléments du type:

    inscrit(Nom_étudiant, Liste_de_cours)

    - *LC*: liste des cours

    la liste *LC* contient des éléments du type:

    epreuve(Cours,Type)
- les listes *FDE* et *FDP* sont créées à partir de ces 3 listes
    - *FDE*: éléments du type

    passe(Nom_étudiant, Cours, Mode, DJ)

    où *DJ* est une variable FD et *Mode* peut prendre les valeurs suivantes:

oral (cas où l'oral se déroule pour tous l'après-midi suivant l'écrit du matin)

ecrit

oral(G) (avec *G* variable FD désignant le n° du groupe)

- *FDP*: éléments du type

  examine(Nom_professeur, Cours, Mode, DJ)

  où *DJ* est une variable FD et *Mode* prend une des valeurs suivantes:

  ecrit,

  oral

  oral(G) avec *G* représentant un numéro de groupe.

Quelques exemples:

passe(dupont, info2208, oral(G), X)

X désigne le numéro du demi-jour correspondant à l'interrogation de *dupont* (faisant partie du groupe *G*) à l'oral du cours d'*info2208*.

La détermination de la valeur de *G* sera confiée au programme.

passe(durand, info2222, ecrit, X)

X désigne le numéro du demi-jour correspondant à l'interrogation écrite de *dupont* pour le cours d'*info2222*.

passe(durand, info2222, oral, X)

X désigne le numéro du demi-jour correspondant à l'interrogation de *dupont* à l'oral commun du cours d'*info2222*.

examine(mno, info2105, oral(5), X)

X désigne le numéro du demi-jour correspondant à l'interrogation du groupe 5 pour l'examen d'*info2105* par *mno*.

3.3.2 Code et commentaires

La procédure *cree_liste* permet de générer les listes *LP*, *LC* et *LE* à partir des fichiers textes adéquats. Les procédures *cree_FDE* et *cree_FDP* génèrent les listes *FDE* et *FDP* à partir de ces dernières.

```
listesFD(FDE,FDP):-
   cree_liste('profs.txt',LP),
   cree_liste('epr.txt',LC),
   cree_liste('inscr.txt',LE),
   cree_FDE(LE,LC,FDE),
   cree_FDP(LP,LE,LC,FDP).
```

Il est difficile de détailler ici tous les modules du programme. Nous vous renvoyons, pour cela, aux annexes dans lesquelles vous trouverez le listing complet d'une des versions, largement commentée, du programme. Nous nous contenterons donc, dans ce chapitre, de donner quelques éclaircissements sur certaines parties plus délicates de ce programme.

Nous détaillons cependant les premiers modules de manière à donner une idée du style de programmation.

La procédure *cree_liste* est évoquée dans le chapitre 5 consacré à la saisie des données. Quant à la procédure *cree_FDE*, elle a besoin de certains paramètres tels le nombre de jours de la session, le nombre maximum de groupes à constituer pour un oral[41] et la liste des congés, c-à-d. des jours de la session pendant lesquels il n'est pas possible d'interroger (dimanches, jours fériés éventuels).

```
cree_FDE(LE,LC,FDE):-
    cree_liste('param.txt',[NDJ,NGMax,_]),
    cree_liste('conges.txt',LCong),
    genere_FDE(LE,LC,NDJ,NGMax,LCong,FDE).
```

Les données nécessaires à la constitution de la liste *FDE* apparaissent progressivement: la liste *LE* des étudiants, la liste *LC* des cours, le nombre de jours de la session *NDJ*, le nombre de groupe maximum *NGMax* et la liste *LCong* des jours de congé.

La procédure *genere_FDE* va agir de manière récursive sur la liste des étudiants. Chaque étudiant est pris en compte avec sa liste de cours par la procédure *gen_etud* qui produira une partie *R1* de la liste *FDE*.

```
genere_FDE([],_,_,_,_,[]):-
    !.

genere_FDE([inscrit(Nom,LCE)|Rinscrits],LC,NDJ,NGMax,LCong,FDE):-
    gen_etud(Nom,LCE,LC,NDJ,NGMax,LCong,R1),
    genere_FDE(Rinscrits,LC,NDJ,NGMax,LCong,R2),
    append(R1,R2,FDE).
```

La procédure *gen_etud* s'occupe donc de générer la partie de la liste *FDE* qui concerne un seul étudiant, celui dont le nom est fourni en argument.

```
gen_etud(_,[],_,_,_,_,[]):-
    !.

gen_etud(Nom,[C|Rcours],LC,NDJ,NGMax,LCong,FDE):-
    member(epreuve(C,Type),LC),
    !,
    gen_etud_ok(Nom,C,Type,NDJ,NGMax,LCong,R1),
    gen_etud(Nom,Rcours,LC,NDJ,NGMax,LCong,R2),
    append(R1,R2,FDE).

gen_etud(Nom,[_|Rcours],LC,NDJ,NGMax,LCong,FDE):-
    gen_etud(Nom,Rcours,LC,NDJ,NGMax,LCong,FDE).
```

Elle fonctionne récursivement sur la liste des cours de l'étudiant et si le cours fait partie de la liste des épreuves, la procédure *gen_etud_ok* est mise en œuvre.

On peut observer comment se traduit, dans ce paradigme de programmation, ce qui serait considéré comme une alternative en programmation impérative.

---

[41] Cette valeur permet de limiter au départ le domaine des variables FD correspondant aux numéros des groupes.

La procédure *gen_etud_ok* va devoir s'occuper des types d'épreuves prévues pour le cours concerné. Elle doit créer autant d'éléments dans la liste *FDE* qu'il y a d'épreuves à passer par l'étudiant pour ce cours (2 au plus).

Elle peut déjà introduire, au niveau des variables, certaines contraintes:

- pas d'examen un jour de conge
- l'écrit précède l'oral
- si l'écrit et l'oral se déroulent le même jour, l'écrit est le matin (demi-jour impair) et l'oral est l'après-midi (demi-jour suivant)

Cette manière de procéder est commode. Plutôt que de placer de telles contraintes dans le module de description des contraintes, on rogne le domaine des variables FD dès leur construction.

```
gen_etud_ok(Nom,C,ecrit*oral,NDJ,_,LCong,
                    [ passe(Nom,C,ecrit,X), passe(Nom,C,oral,Y) ]):-
   !,
   fd_domain(X,1,NDJ),
   fd_domain(Y,1,NDJ),
   impair(X,1,NDJ),
   Y#=#X+1,
   pas_conge(X,LCong),
   pas_conge(Y,LCong).

gen_etud_ok(Nom,C,ecrit,NDJ,_,LCong,[ passe(Nom,C,ecrit,X) ]):-
   !,
   fd_domain(X,1,NDJ),
   pas_conge(X,LCong).

gen_etud_ok(Nom,C,ecrit+oral(_),NDJ,NGMax,LCong,
                [ passe(Nom,C,ecrit,X), passe(Nom,C,oral(G),Y) ]):-
   !,
   fd_domain(X,1,NDJ),
   fd_domain(Y,1,NDJ),
   fd_domain(G,1,NGMax),
   X#<#Y,
   pas_conge(X,LCong),
   pas_conge(Y,LCong).

gen_etud_ok(Nom,C,oral(_),NDJ,NGMax,LCong,
                    [ passe(Nom,C,oral(G),X) ]):-
   !,
   fd_domain(X,1,NDJ),
   fd_domain(G,1,NGMax),
   pas_conge(X,LCong).
```

À ce niveau, on construit véritablement la liste puisque des éléments de type *passe (...)* y sont ajoutés. Le prédicat prédéfini *fd_domain* permet de fixer le domaine des variables créées et des contraintes sont déjà introduites comme, par exemple:

- *X#<#Y* qui traduit le fait que l'écrit précède l'oral,
- *pas_conge(X,LCong)* qui impose à *X* de ne pas prendre valeur dans la liste des jours de congé.

La procédure *pas_conge* demande peu de commentaires.

```
pas_conge(_,[]):-
    !.

pas_conge(X,[Y|L]):-
    X#\=#Y,
    pas_conge(X,L).
```

La procédure *impair* force la variable à prendre une valeur impaire dans le domaine. Cette fonction est nécessaire si on veut qu'un écrit ait lieu le matin et l'oral de la même épreuve, l'après-midi. Son code qui suit est élémentaire.

```
impair(_,VI,VF):-
    VI+1>VF,
    !.

impair(X,VI,VF):-
    X #\=#VI+1,
    impair(X,VI+2,VF).
```

À noter encore que les symboles « \= » marquent la différence et que le symbole « # » qui les entoure indique au solveur de pratiquer la consistance totale aux arcs.

Voilà donc développée, de manière plus ou moins complète, une des branches d'un des modules du programme. Ce développement illustre la manière donc le programme fonctionne et nous pensons qu'il n'est pas nécessaire de développer chacun des modules de la sorte. Le listing complet du programme figure de toute façon en annexe. Dans la suite de ce chapitre, nous nous contenterons de développer les modules un peu plus particuliers.

La procédure *cree_FDP* fonctionne globalement de la même manière. Elle a également besoin des mêmes paramètres. Une procédure *genere_FDP* parcourt la liste des professeurs de manière récursive. La procédure *gen_prof* se charge alors de traiter la liste des cours du professeur concerné et la procédure *gen_prof_ok* ne traite que les cours qui font l'objet d'examens et crée progressivement la liste FDP. Voici le code de cette procédure:

```
gen_prof_ok(Nom,C,ecrit*oral,_,_,NDJ,_,LCong,
            [examine(Nom,C,ecrit,X),examine(Nom,C,oral,Y)]):-
    fd_domain(X,1,NDJ),
    pas_conge(X,LCong),
    impair(X,1,NDJ),
    fd_domain(Y,1,NDJ),
    pas_conge(Y,LCong),
    Y#=#X+1.
```

```
gen_prof_ok(Nom,C,ecrit+oral(N),LE,_,NDJ,_,LCong,
            [examine(Nom,C,ecrit,X)|FDP]):-
   fd_domain(X,1,NDJ),
   pas_conge(X,LCong),
   combien(C,LE,M),
   G is ceiling(M/N),
   gen_prof_ok_oralecrit(G,X,Nom,C,NDJ,LCong,FDP).

gen_prof_ok(Nom,C,oral(N),LE,_,NDJ,_,LCong,FDP):-
   combien(C,LE,M),
   G is ceiling(M/N),
   gen_prof_ok_oral(G,Nom,C,NDJ,LCong,FDP).

gen_prof_ok(Nom,C,ecrit,_,_,NDJ,_,LCong,[examine(Nom,C,ecrit,X)]):-
   fd_domain(X,1,NDJ),
   pas_conge(X,LCong).
```

Les deuxième et troisième définitions demandent un petit commentaire. La procédure *combien* calcule le nombre *M* d'étudiants inscrits au cours *C* dans la liste *LE*. Le nombre de groupes résulte alors d'une simple division, arrondie à l'unité supérieure (*ceiling*), de ce nombre par le nombre d'étudiants interrogés sur un demi-jour.

Dans chacun de ces cas, le relais est passé à une autre procédure *gen_prof_ok_oralecrit* et *gen_prof_ok_oral*. Leur principe est de créer autant d'éléments que de groupes pour les oraux et un élément pour les écrits. Comme pour les éléments de *FDE*, quelques contraintes sont incluses. Voici, pour l'exemple, le code de *gen_prof_ok_oralecrit*.

```
gen_prof_ok_oralecrit(0,_,_,_,_,_,[]):-
   !.

gen_prof_ok_oralecrit(G,X,Nom,C,NDJ,LCong,
                      [examine(Nom,C,oral(G),Y)|RFDE]):-
   G1 is G-1,
   fd_domain(Y,1,NDJ),
   pas_conge(X,LCong),
   Y#>#X,
   gen_prof_ok_oralecrit(G1,X,Nom,C,NDJ,LCong,RFDE).
```

La valeur de *G* diminue à chaque passage (arrêt à $G = 0$). *X* est la variable FD correspondant à l'écrit du même cours, ce qui justifie l'inégalité (oral après écrit).

3.4 Génération des contraintes

3.4.1 Description

Les deux listes *FDE* et *FDP* étant créées, il va falloir les utiliser pour exprimer les autres contraintes. Les contraintes prises en compte sont les suivantes:

- Les professeurs sont indisponibles certains jours ou demi-jours (*prof_dispo*).
- Les groupes sont constitués d'un nombre égal d'étudiants (à une unité près) et chaque étudiant est associé à un seul groupe pour une épreuve donnée (*attrib_groupes*)[42].
- Les étudiants passent l'écrit d'un même cours au même moment (*prof_etud*). Les étudiants d'un même groupe d'oral sont également présents au même moment. Dans chacun de ces cas, le professeur concerné est présent.
- Il est souhaitable que les différents examens d'un même étudiant soient distants de quelques demi-jours (*dist_exam*). Cette valeur est lue dans un fichier de configuration nommé *param.txt*.
- Un professeur ne peut interroger deux groupes au même moment (*pas2groupes*).
- Un examen oral ne peut avoir lieu en même temps qu'un écrit, qu'un oral commun ou qu'un oral ne comprenant qu'un seul groupe (*pas_ecrit_oral*)[43].

3.4.2 Code et commentaires

Voici le code du module principal:

```
contraintes(FDE,FDP):-
    cree_liste('indis83.txt',LI),
    prof_dispo(FDP,LI),
    cree_liste('inscr.txt',LE),
    cree_liste('epr.txt',LC),
    attrib_groupes(FDE,LC,LE),
    prof_etud(FDE,FDP),
    cree_liste('param.txt',[_,_,D]),
    dist_exam(FDE,D),
    pas2groupes(FDP).
```

Les listes nécessaires sont à nouveau reconstituées:

- la liste *LI* des indisponibilités des professeurs dont un élément typique est *indispo(Nom_prof, Liste_de_demi_jours)*
- la liste *LC* des cours dont un élément typique est *epreuve(Cours, Type)*
- la liste *LE* des étudiants dont un élément typique est *inscrit(Nom_etudiant, Liste_de_cours)*

Nous ne développons pas l'expression de certaines contraintes car leur code ne comporte pas de réelles difficultés de compréhension. Nous avons choisi de décrire les procédures *prof_etud* et *attrib_groupes*.

La procédure *prof_etud* assure que les professeurs sont présents à tous leurs examens avec les étudiants concernés. Elle est récursive sur la liste *FDP*.

```
prof_etud(_,[]):-
    !.
```

---

[42] Cette contrainte n'est pas prise en compte dans la version "groupes anonymes" mais bien dans la version "horaires individuels".

[43] Cette contrainte est inutile dans la version "horaires individuels" puisqu'un étudiant ne peut avoir deux examens le même jour.

```
prof_etud(FDE,[examine(_,C,ecrit,X)|RFDP]):-
   !,
   prof_etud_ens(FDE,examine(_,C,ecrit,X)),
   prof_etud(FDE,RFDP).

prof_etud(FDE,[examine(_,C,oral,X)|RFDP]):-
   !,
   prof_etud_ens(FDE,examine(_,C,oral,X)),
   prof_etud(FDE,RFDP).

prof_etud(FDE,[examine(_,C,oral(N),X)|RFDP]):-
   !,
   prof_etud_ens(FDE,examine(_,C,oral(N),X)),
   prof_etud(FDE,RFDP).

prof_etud(FDE,[examine(_,_,_,_)|RFDP]):-
   prof_etud(FDE,RFDP).
```

Pour chaque élément de la liste *FDP*, un parcours de la liste FDE est effectué pour égaliser les variables qui doivent l'être. C'est la procédure *prof_etud_ens* dont le code suit qui effectue ce travail d'égalisation.

```
prof_etud_ens([],_):-
   !.

prof_etud_ens([passe(_,C,ecrit,X1)|RFDE],examine(_,C,ecrit,X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,ecrit,X)).

prof_etud_ens([passe(_,_,_,_)|RFDE],examine(_,C,ecrit,X)):-
   !,
   prof_etud_ens(RFDE,examine(_,C,ecrit,X)).

prof_etud_ens([passe(_,C,oral,X1)|RFDE],examine(_,C,oral,X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,oral,X)).

prof_etud_ens([passe(_,C,oral(N),X1)|RFDE],examine(_,C,oral(N),X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,oral(N),X)).
```

```
prof_etud_ens([passe(_,_,_,_)|RFDE],examine(_,C,T,X)):-
    !,
    prof_etud_ens(RFDE,examine(_,C,T,X)).
```

Tous les cas sont envisagés. L'égalité n'est évidemment requise que lorsqu'il y a concordance entre les cours et les types, voire les numéros de groupes.

La procédure *attrib_groupes* est particulière en ce sens que c'est elle qui va attribuer un groupe à chaque étudiant. Cette procédure va parcourir récursivement la liste des cours en s'intéressant à ceux qui comportent un oral. Elle passera le relais à une procédure *attribue* qui fixera la valeur de *G* pour chaque élément de la liste *FDE* du type *passe(*, *, oral(G), X)*. De la sorte, chaque étudiant fera partie d'un groupe pour chacun de ses oraux.

Voici le code de la procédure *attrib_groupes*:

```
attrib_groupes(_,[],_):-
    !.

attrib_groupes(FDE,[epreuve(C,ecrit+oral(N))|Rcours],LE):-
    !,
    calcule(LE,N,C,NEG,NGMax,NG),
    attribue(C,NEG,NGMax,NG,0,FDE),
    attrib_groupes(FDE,Rcours,LE).

attrib_groupes(FDE,[epreuve(C,oral(N))|Rcours],LE):-
    !,
    calcule(LE,N,C,NEG,NGMax,NG),
    attribue(C,NEG,NGMax,NG,0,FDE),
    attrib_groupes(FDE,Rcours,LE).

attrib_groupes(FDE,[epreuve(_,_)|Rcours],LE):-
    attrib_groupes(FDE,Rcours,LE).
```

La procédure *calcule* que nous ne détaillons pas fournit notamment comme valeurs, le nombre d'étudiants à placer dans les groupes ainsi que le nombre de groupes supérieurs aux autres d'une unité.

Exemple: 30 étudiants, 8 par demi-jour ⇒ 4 groupes: 2 de 8 et 2 de 7 ⇒ valeurs renvoyées: 8 et 2

La procédure *attribue* parcourt la liste *FDE*. La valeur de G est décrémentée chaque fois que le nombre d'étudiants est atteint. Ce nombre d'étudiants diminue d'une unité lorsque le nombre de groupes supérieurs est atteint.

```
attribue(_,_,_,0,_,_):-
    !.

attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-2,
    CE<N,
    CG>NGMax,
    !,
```

```
    CE1 is CE+1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG,CE1,RFDE).
attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-2,
    CE=N,
    CG>NGMax,
    !,
    CE1 is 0,
    CG1 is CG-1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG1,CE1,RFDE).
attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-1,
    CE<N,
    CG=<NGMax,
    !,
    CE1 is CE+1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG,CE1,RFDE).
attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-1,
    CE=N,
    CG=<NGMax,
    !,
    CG1 is CG-1,
    CE1 is 0,
    G#=#CG,
    attribue(C,NEG,NGMax,CG1,CE1,RFDE).
attribue(C,NEG,NGMax,CG,CE,[passe(_,_,_,_)|RFDE]):-
    attribue(C,NEG,NGMax,CG,CE,RFDE).
```

3.5 Labélisation

3.5.1 Description

Il s'agit de fixer, dans ce module, la manière dont les variables sont déterminées lorsque les contraintes sont exprimées. Rappelons que la labélisation inclut deux types de choix:

- dans quel ordre faut-il fixer les variables?
- pour une variable donnée, quelle valeur choisir parmi celles qui sont encore disponibles?

Plusieurs stratégies sont évidemment possibles et nous en décrivons quelques-unes dans le chapitre 4 consacré aux expérimentations. Au niveau de ce paragraphe, nous nous contenterons d'illustrer une réponse possible à ces deux questions.

Dans l'exemple choisi, ce sont d'abord les variables correspondant aux professeurs qui sont fixées en commençant par ceux dont les disponibilités sont les moindres.

Pour les variables correspondant aux étudiants, il n'y a pas de stratégie particulière si ce n'est que les valeurs sont choisies de manière aléatoire.

3.5.2 Code et commentaires

Le code du module général fait donc apparaître une procédure pour la labélisation des variables pour les professeurs (*label_prof*) et une procédure pour la labélisation des variables pour les étudiants (*label_etud*).

```
label(FDE,FDP):-
    label_prof(FDP),
    label_etud(FDE).
```

Examinons la première des deux procédures. Avant de fixer les valeurs, les professeurs sont classés par ordre décroissant d'indisponibilité. La liste *FDP* est transformée en une liste *FDP1* dont les éléments ont la forme *(N, examine(Nom,C,G,X))* ou *N* représente le nombre de valeurs possibles pour *X* (procédure *size_p_list*).

Cette nouvelle liste est triée en *FDP2* sur base de la clé qui est ce nombre de jours (procédure prédéfinie *keysort*). Ce sont les valeurs de *FDP2* qui seront labélisées (procédure *label_pr*).

```
label_prof(FDP):-
    size_p_list(FDP,FDP1),
    keysort(FDP1,FDP2),
    label_pr(FDP2).
```

La procédure *size_p_list* prépare la liste au triage grâce au prédicat prédéfini *keysort*.

```
size_p_list([],[]):-
    !.

size_p_list([examine(Nom,C,G,X)|R],[(N,examine(Nom,C,G,X))|Rs]):-
    fd_size(X,N),
    size_p_list(R,Rs).
```

La procedure *label_pr* fixe les valeurs des variables concernant les professeurs en choisissant aléatoirement les valeurs parmi les valeurs possibles.

```
label_pr([]):-
    !.

label_pr([(_,examine(_,_,_,X))|R]):-
    fd_labeling(X,[value_method(random)]),
    label_pr(R).
```

La seconde des deux procédures est ici ramenée à sa plus simple expression. En première session, la plupart des étudiants ayant un nombre d'épreuves sensiblement égal à passer, un classement se justifie moins[44].

```
label_etud([]):-
    !.

label_etud([passe(_,_,_,X)|RFDE]):-
    fd_labeling(X,[value_method(random)]),
    label_etud(RFDE).
```

Dans cet exemple, nous avons à nouveau opté pour un choix aléatoire des valeurs.

3.6 Impression

3.6.1 Description

Cette partie représente un nombre important de lignes de code relativement peu intéressantes si ce n'est quelques passages spécifiques qui montrent comment il est possible de générer des fichiers destinés à être relus (fichiers textes ou documents HTML essentiellement dans ce contexte). Nous nous limiterons donc à illustrer de tels passages pour éviter une débauche de code.

Dans le module principal, on trouve deux procédures, l'une créant des fichiers textes de contrôle pour le débogage, l'autre créant les fichiers destinés à être réutilisés. Nous n'avons pas supprimé la première procédure à des fins d'illustration.

3.6.2 Code et commentaires

Voici le code du module principal:

```
impressions(FDE,FDP):-
    controle(FDE,FDP),
    imprime(FDE,FDP).
```

Le module *controle* se contente d'imprimer les listes *FDE* et *FDP*, élément par élément, dans un fichier *resultats.txt*.

```
controle(FDE,FDP):-
    open('resultats.txt',write,S),
    affiche(FDE,S),
    affiche(FDP,S),
    close(S2).
```

La procédure *affiche* parcourt récursivement la liste qui lui est fournie. Le rôle de *S*, flux de données logique, est expliqué dans le chapitre 5.

```
affiche([],_):-
    !.
```

---

[44] Ce n'est pas le cas pour une seconde session ou il vaut mieux commencer par les étudiants qui ont un maximum d'épreuves à passer.

```
affiche([X|L],S):-
      write(S,X),
      write(S,'<BR>'),
      affiche(L,S).
```

La seconde procédure du bloc d'impression est celle qui nous intéresse davantage. Elle génère des documents au format HTML qui, lorsqu'ils seront interprétés par un navigateur, fourniront l'horaire général et les horaires individuels si cette dernière version est utilisée.

La procédure exploite donc les informations disponibles dans les listes *FDE* et *FDP* pour créer un fichier nommé *resultats.htm* et un fichier *hor_ind.htm* qui soient interprétables par un navigateur web.

L'affichage par le navigateur produit pour *resultats.htm*, un tableau qui met en parallèle les examens des trois années d'étude dans l'ordre chronologique (une ligne = 1/2 jour). Pour *hor_ind.htm*, on aura une succession de tableaux reprenant les horaires individuels des étudiants (n'apparaîtront que les demi-jours ou l'étudiant doit être interrogé.

```
imprime(FDE,FDP):-
   impr_horaire(FDP),
   impr_hor_ind(FDE).
```

Pour la création du fichier contenant l'horaire global,

```
impr_horaire(FDP):-
   cree_liste('dates.txt',LD),
   open('resultats.htm',write,S),
   debut_tab(S),
   horaire(FDP,LD,S),
   fin_tab(S),
   close(S).
```

La procédure *debut_tab* crée l'entête, la procédure *horaire* crée le corps du tableau et la procédure *fin_tab* le termine correctement.

Le fichier *LD* des correspondances entre les valeurs et les dates doit être généré, de même que le fichier des étudiants, *LE*.

```
impr_hor_ind(FDE):-
   cree_liste('dates.txt',LD),
   cree_liste('inscr.txt',LE),
   open('hor_ind.htm',write,S),
   hor_etud(FDE,LE,LD,S),
   close(S).
```

Le document contiendra un tableau pour chaque étudiant. L'entête de tableau se trouvera donc décrite dans la procédure *hor_etud* qui parcourra récursivement la liste *LE*.

La description de l'impression pour l'horaire étant assez longue principalement en raison de la nécessité d'un affichage en trois colonnes, nous nous contenterons d'illustrer ici l'impression des horaires individuels.

```
hor_etud(_,[],_,_):-
    !.
hor_etud(FDE,[inscrit(Nom,_)|Rinscrits],LD,S):-
    debut_tab(S),
    write(S,Nom),
    changeligne(S),
    result(FDE,Nom,LD,S),
    fin_tab(S),
    write(S,'<HR>'),
    hor_etud(FDE,Rinscrits,LD,S).
```

Les procédures *debut_tab*, *fin_tab* et *changeligne* ont comme but d'envoyer les caractères correspondant aux balises HTML adéquates dans le fichier (cfr annexes).

La procédure *result* mérite un examen plus approfondi.

La procédure crée et envoie dans le fichier HTML les données correspondant à l'étudiant *Nom*. Une liste est créée avec les épreuves de l'étudiant dont un élément type est *interro(X,C,G)*. Cette liste est ensuite triée avant enregistrement des informations dans le fichier.

```
result(_,[],_,_):-
    !.

result(FDE,Nom,LD,S):-
    findall(interro(X,C,G),member(passe(Nom,C,G,X),FDE),LX),
    sort(LX,LY),
    enregistre(LY,LD,S).
```

Les procédures *findall* et *sort* sont des prédicats prédéfinis. Enfin, la procédure *enregistre* crée véritablement le contenu du document HTML. Voici le code de cette procédure. Elle parcourt récursivement la liste triée en écrivant pour chaque élément, la date extraite du fichier *LD*, le cours et le type d'examen (en ce compris le groupe si c'est un oral).

```
enregistre([],_,_):-
    !.

enregistre([interro(X,C,G)|Rinterro],LD,S):-
    member(date(X,Date),LD),
    !,
    write(S,Date),
    changecellule(S),
    write(S,C),
    changecellule(S),
    write(S,G),
    changeligne(S),
    enregistre(Rinterro,LD,S).
```

Chapitre 4 EXPÉRIMENTATIONS ET RÉSULTATS

Il ne faut cesser de s'enfoncer dans sa nuit: c'est alors que brusquement, la lumière se fait.
(Francis Ponge)

Les expérimentations dont il est question ici ne sont pas reprises par ordre chronologique de leur développement mais elles sont organisées de manière à présenter diverses variantes du programme qui fonctionnent, en développant leurs caractéristiques. Quant aux résultats, il est difficile d'aligner dans cette partie et pour analyse plusieurs horaires produits, d'autant que chacun ne peut être examiné que dans sa globalité. Nous en avons présenté peu. Vous en trouverez plusieurs dans les annexes.

4.1 La version « groupes anonymes »

Cette version a été notamment testée dans les conditions de la session de juin 2003 pour l'ensemble des années de licence et maîtrise. Elle tient compte du nombre d'étudiants interrogés par les professeurs au cours d'un demi-jour d'oral. Ce nombre sert à calculer, à partir du nombre d'étudiants inscrits à un examen, le nombre de groupes à constituer pour l'oral. Un demi-jour est donc prévu dans l'horaire pour chacun des groupes d'un même examen oral. Les étudiants sont amenés à s'inscrire, voire à s'organiser entre eux en cas de difficulté, pour que les groupes soient constitués en tenant compte du nombre maximum d'étudiants interrogés[45].

Cette version possède elle-même quelques variantes que nous allons proposer en en présentant certains résultats. Ces variantes se distinguent essentiellement par la manière dont la labélisation (fixation des valeurs des variables) est organisée et par les heuristiques utilisées pour choisir ces valeurs parmi les valeurs encore possibles. Notre but est de montrer qu'en agissant sur ces deux composantes, il est possible de « corriger le tir » pour un horaire qui serait considéré comme « pas très bon »[46] essentiellement pour des raisons de mauvaises répartitions des épreuves dans la session.

Pour agir sur la qualité des horaires produits, il y a plusieurs pistes possibles. Soit, on se rend compte que des contraintes supplémentaires doivent être définies, soit on tâche d'agir sur la manière dont les valeurs sont fixées. La seconde technique, plus souple, est réalisée essentiellement par l'emprunt de deux voies possibles: le choix d'un ordre dans les variables à fixer et, une fois ce choix effectué, le choix d'une valeur dans l'ensemble de celles qui peuvent encore l'être. On parle à ce propos de **labélisation** et d'**heuristiques**.

4.1.1 Variante n°1

Labélisation

Dans cette version, les groupes étant anonymes, il n'y a pas de labélisation particulière en ce qui concerne les variables liées aux étudiants. Pour rappel, ces variables apparaissent dans des expressions du type:

passe(Nom, Cours, Type, X)

---

[45] Selon nos informations, c'est à peu près comme cela que les choses se passent dans la réalité. Notez toutefois que la version que nous présentons par la suite répartit les étudiants de manière égale dans les groupes, ce qui autorise la production d'un horaire individuel pour chaque étudiant.

[46] Toutes ces expressions entre guillemets traduisent à souhait la subjectivité qui entoure les critères dans la constitution des horaires, que celle-ci soit manuelle ou automatique.

En revanche, il y a labélisation des variables liées aux professeurs selon le principe suivant:

- priorité aux écrits;
- (pour les oraux) priorité aux variables lies aux professeurs dont le nombre de valeurs disponibles est le plus faible.

Heuristique

L'heuristique choisie est celle d'un choix aléatoire des variables. Elle se traduit à deux endroits dans le code de la manière suivante:

```
fd_labeling(X,[value_method(random)])
```

Partie de code commentée

La labélisation se décompose en labélisation des épreuves écrites et labélisation des oraux sur base de la disponibilité résultante des professeurs.

Le suffixe *v1* fait référence à la version « groupes anonymes » et le suffixe *v11* à sa première variante.

```
label_v1(FDP):-
    label_v1_ecrit(FDP),
    label_v1_prof(FDP).
```

La labélisation des écrits se fait en parcourant successivement les éléments de la liste *FDP*[47] et en fixant aléatoirement les variables de ceux qui correspondent à des épreuves écrites.

```
label_v1_ecrit([]):-
    !.
label_v1_ecrit([examine(_,_,ecrit,X)|RFDP]):-
    !,
    fd_labeling(X,[value_method(random)]),
    label_v1_ecrit(RFDP).
```

Les autres sont ignorés.

```
label_v1_ecrit([examine(_,_,_,_)|RFDP]):-
    label_v1_ecrit(RFDP).
```

La labélisation des oraux comprend la création d'une liste ayant comme index le nombre de valeurs encore disponibles pour la variable FD de chaque élément, un tri de la liste *FDP* par ordre croissant de cet index et la labélisation proprement dite.

```
label_v1_prof(FDP):-
    size_prof_list(FDP,FDP1)
    keysort(FDP1,FDP2),
    label_v11_pr(FDP2).
```

La création de la liste indexée se fait sur base d'un parcours de la liste *FDP* en constituant une nouvelle liste prête à l'emploi du prédicat prédéfini de tri sur base d'une clé d'index (*keysort*).

```
size_prof_list([],[]):-
```

---

[47] ...dont pour rappel, les éléments sont du type *examine(Nom, Cours, Type, X)*

```
    !.

size_prof_list([examine(Nom,C,G,X)|R],[(N,examine(Nom,C,G,X))|Rs]):-
    fd_size(X,N),
    size_prof_list(R,Rs).
```

Le prédicat *fd_size* est prédéfini et renvoie le nombre de valeurs *N* encore disponibles pour la variable *X*. La labélisation procède de la même manière que pour les écrits sauf que dans ce cas, seuls les éléments correspondant à des oraux sont pris en compte.

```
label_v11_pr([]):-
    !.

label_v11_pr([(_,examine(_,_,oral(_),X))|RFDP]):-
    !,
    fd_labeling(X,[value_method(random)]),
    label_v11_pr(RFDP).

label_v11_pr([(_,examine(_,_,_,_))|RFDP]):-
    label_v11_pr(RFDP).
```

Problème particulier

Les groupes étant anonymes, il importe de fournir aux étudiants les moyens de s'inscrire dans un groupe sachant qu'ils ne peuvent avoir deux examens au cours de la même matinée. Dès lors, comme il n'est pas tenu compte des étudiants mais seulement des groupes, les épreuves pour lesquelles il n'y a qu'un seul groupe à former doivent être considérées comme les écrits ou comme des oraux communs à tous ceux qui passent l'épreuve[48]. Elles ne peuvent être organisées en même temps que d'autres épreuves.

Notons qu'il est possible, sans beaucoup d'effort de programmation, d'étendre cette exigence aux groupes des épreuves orales comptant moins de *N* groupes (*N*=3 par exemple) pour garantir un meilleur choix aux étudiants.

Cette contrainte est prise en compte par le prédicat supplémentaire *pas_ecrit_oral*.

```
contraintes(FDE,FDP):-
    cree_liste('indis.txt',LI),
    prof_dispo(FDP,LI),
    prof_etud(FDE,FDP),
    cree_liste('param.txt',[_,_,D]),
    dist_exam(FDE,D),
    pas2groupes(FDP),
    pas_ecrit_oral(FDP,FDP).
```

La procédure *pas_ecrit_oral* parcourt la liste *FDP*, élément par élément

```
pas_ecrit_oral([],_):-
    !.
```

[48] ...et qui servent essentiellement d'oral de rattrapage

```
pas_ecrit_oral([E|RFDP],FDP):-
    !,
    pas_oral(E,RFDP,FDP),
    pas_ecrit_oral(RFDP,FDP).
```

La procédure *pas_oral* parcourt les éléments qui suivent l'élément donné dans la liste en enregistrant les impossibilités de coexistence d'épreuves sauf si celles-ci sont des oraux pour lesquels plusieurs groupes sont formés.

```
pas_oral(_,[],_):-
    !.

pas_oral(examine(_,C,oral(M),X),[examine(_,_,oral(N),_)|RFDP],FDP):-
    M\=1,
    N\=1,
    !,
    pas_oral(examine(_,C,oral(M),X),RFDP,FDP).

pas_oral(examine(_,C,oral(1),X),[examine(_,_,oral(N),_)|RFDP],FDP):-
    N\=1,
    member(examine(_,C,oral(2),_),FDP),
    !,
    pas_oral(examine(_,C,oral(1),X),RFDP,FDP).

pas_oral(examine(_,C1,oral(1),X),[examine(_,C2,oral(1),_)|RFDP],FDP)
:-
    member(examine(_,C1,oral(2),_),FDP),
    member(examine(_,C2,oral(2),_),FDP),
    !,
    pas_oral(examine(_,C1,oral(1),X),RFDP,FDP).

pas_oral(examine(_,C1,oral(M),X),[examine(_,C2,oral(1),_)|RFDP],FDP)
:-
    M\=1,
    member(examine(_,C2,oral(2),_),FDP),
    !,
    pas_oral(examine(_,C1,oral(M),X),RFDP,FDP).

pas_oral(examine(_,C,T,X),[examine(_,_,_,Y)|RFDP],FDP):-
    X#\=#Y,
    pas_oral(examine(_,C,T,X),RFDP,FDP).
```

Comme on peut le constater, le programme doit être attentif à la « non-existence » d'un deuxième groupe pour un oral donné.

Résultats

Le solveur produit de nombreuses solutions parmi lesquelles un choix est possible. La génération d'une nouvelle solution est commode vu le temps mis à la produire (de l'ordre de 2 ou 3 secondes). L'utilisation d'une heuristique basée sur le choix d'une valeur aléatoire produit des solutions consécutives relativement différentes les unes des autres[49].

Voici un des résultats produits.

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM	info2109:oral(2)		
31-mai PM		info2233:oral(4)	info2327:oral(1)
1-juin AM			
1-juin PM			
2-juin AM			info2325:oral(1)
2-juin PM	info2114:oral(1)		
3-juin AM		info2208:oral(5)	
3-juin PM		info2231:oral(1)	
4-juin AM	ielv2113:oral(1)		
4-juin PM		info2232:ecrit	
5-juin AM	info2109:oral(3)	info2208:oral(4)	info2324:oral(2)
5-juin PM		info2233:oral(6)	info2322:oral(2)
6-juin AM	info2102:oral(1)		
6-juin PM		info2204:ecrit	
7-juin AM	info2110:oral(1)		
7-juin PM		info2205:oral(1)	
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM		info2230:oral(1)	
10-juin PM		info2223:oral(1)	
11-juin AM		info2211:ecrit	
11-juin PM		ielv2212:oral(1)	
12-juin AM		info2222:ecrit	
12-juin PM		info2221:oral(1)	
13-juin AM	info2106:ecrit		
13-juin PM		info2233:oral(5)	
14-juin AM			info2326:oral(1)
14-juin PM	info2109:oral(4)		
15-juin AM			
15-juin PM		info2211:oral(1)	
16-juin AM		info2202:ecrit	
16-juin PM		info2202:oral	
17-juin AM			info2301:ecrit

[49] Comme nous le verrons dans les perspectives, ce n'est pas toujours une bonne chose dans la mesure où une solution proposée peut s'avérer bonne à 80%, par exemple.

17-juin PM	info2112:oral(1)		
18-juin AM		info2233:oral(3)	
18-juin PM	info2105:ecrit		
19-juin AM	info2105:oral(2)		info2301:oral(2)
19-juin PM	info2109:oral(1)	info2208:oral(2)	info2301:oral(1)
20-juin AM			ielv2307:oral(1)
20-juin PM	info2105:oral(4)		info2322:oral(1)
21-juin AM		info2233:oral(2)	
21-juin PM	info2111:ecrit		
22-juin AM			
22-juin PM			info2301:oral(4)
23-juin AM	info2105:oral(3)		info2324:oral(1)
23-juin PM	info2105:oral(1)	info2233:oral(1)	
24-juin AM			info2302:oral(1)
24-juin PM	info2108:ecrit		
25-juin AM		info2208:oral(1)	
25-juin PM		sent2288:ecrit	
26-juin AM	info2111:oral(2)	info2208:oral(3)	info2327:oral(2)
26-juin PM		info2211:oral(2)	
27-juin AM		info2209:ecrit	
27-juin PM			info2327:oral(3)
28-juin AM	info2111:oral(3)		info2322:oral(3) info2301:oral(3)
28-juin PM		info2211:oral(3)	
29-juin AM			
29-juin PM	info2105:oral(5) info2111:oral(1)		
30-juin AM	info2111:oral(4)		info2301:oral(5) info2326:oral(2)
30-juin PM	info2107:ecrit		

On trouvera, en annexe, d'autres exemples de solutions que l'on pourra comparer. On observera que les épreuves concernant les oraux ne comptant qu'un seul groupe et les oraux communs ne sont jamais en parallèle avec d'autres épreuves.

Il est évidemment difficile de juger de la qualité d'un tel horaire sans avoir une idée de ce que sont ou pourraient être les horaires individuels des étudiants. Une concentration d'épreuves dans l'horaire général ne correspond en effet pas nécessairement à une concentration d'épreuves pour les étudiants puisque l'horaire concerne tous les groupes. C'est la raison pour laquelle une version produisant des horaires individuels est largement souhaitable.

4.1.2 Variante n°2

Labélisation

Dans cette variante, nous gardons le même type de labélisation.

Heuristique

Nous choisissons une heuristique basée sur le choix de la plus petite valeur disponible pour ce qui est des examens écrits et des oraux.

```
fd_labeling(X,[value_method(min)])
```

Résultats

Le solveur produit une solution dans un délai semblable.

On trouve, en annexe, des exemples de solutions que l'on peut comparer. Assez logiquement, on observe que les épreuves écrites sont concentrées dans la toute première partie de la session et que les oraux se bousculent dans la première moitié de cette session. Ces résultats, peu intéressants, sont la conséquence du choix de l'heuristique.

4.1.3 Variante n°3

Labélisation

Dans cette variante, nous gardons le même type de labélisation.

Heuristique

Nous gardons l'heuristique basée sur le choix de la plus petite valeur possible **uniquement pour les épreuves écrites**.

Pour les oraux, nous conservons l'heuristique du choix aléatoire pour éviter une trop forte concentration des épreuves dans la première partie de la session.

Résultats

On trouve, en annexe, un exemple de solution. On observe que les épreuves écrites sont concentrées dans la toute première partie de la session et que les oraux sont répartis assez correctement. On se rapproche de ce que nous pourrions appeler un « bon horaire ». Un élément n'est cependant pas pris en compte, c'est la relative « difficulté » que représente chacune des épreuves. Ce point est abordé dans les perspectives.

4.1.4 Autres variantes possibles

L'analyse des solutions montre que les contraintes définies sont bien respectées. Une observation cependant: les oraux du même cours d'un professeur ne sont pas toujours très concentrés. Il n'est pas du tout sûr que cette concentration soit un souhait[50]. S'il fallait rencontrer le problème, on voit difficilement comment une heuristique ou une labélisation particulière pourraient renforcer cette concentration.

Les heuristiques disponibles correspondent, outre au choix de la plus petite des valeurs (choix par défaut), au choix d'une valeur au hasard, de la plus grande des valeurs, à un choix qui s'effectue des bornes vers la valeur centrale ou de la valeur centrale vers les bornes. Aucune d'entre elles ne peut garantir un groupement des épreuves puisqu'il s'agit de relations entre différentes variables FD.

La solution passe par l'écriture d'une contrainte supplémentaire fixant, par exemple, l'écart maximum entre deux groupes consécutif d'un même oral. C'est possible puisque ces valeurs se retrouvent dans des éléments consécutifs de la liste *FDP*.

```
[examine(ble,info2109,oral(3),26),
 examine(ble,info2109,oral(2),12),
 examine(ble,info2109,oral(1),10),
 examine(clo,info2102,oral(4),22),
 examine(clo,info2102,oral(3),36),
 examine(clo,info2102,oral(2),35),
 examine(clo,info2102,oral(1),26),...]
```

---

[50] ...selon les informations dont nous disposons

Symboliquement:

[..., examine(Nom, Cours, oral(G),X), examine(Nom, Cours, oral(H), Y),...] $\Rightarrow X <= Y + N$

où *N* représente un écart maximum entre deux groupes du même oral.

4.2 La version « horaires individuels »

Demander aux étudiants de se répartir dans des groupes pour lesquels un horaire de passage est prévu peut s'avérer intéressant en ce sens que l'étudiant peut choisir les groupes qui lui conviennent le mieux en fonction de la planification qu'il souhaite de sa session d'examens. L'inconvénient, c'est que des conflits peuvent très vite surgir dès l'instant où le nombre de places par groupe est limité. Il est même probable que des étudiants se retrouvent dans l'impossibilité de choisir un groupe pour un cours donné. Ils doivent alors prendre des accords avec d'autres étudiants pour échanger, voire contacter la secrétaire qui gère alors le problème « en bonne mère de famille »[51].

Pour éviter ce genre de problème, il est souhaitable de répartir les étudiants dans des groupes. La répartition en groupes offre également d'autres avantages. Elle permet d'exploiter à fond les possibilités de placement pour les groupes. En effet, alors que dans la version « groupes anonymes », il fallait veiller à ce que les épreuves orales des cours qui ne comptent qu'un seul groupe et d'autres épreuves ne soient pas simultanées, la contrainte disparaît ici car les vérifications se font au niveau des activités des étudiants.

La possibilité de produire, pour chaque étudiant, un horaire individuel est très appréciable en ce qu'elle permet de se rendre compte assez facilement de la qualité de l'horaire[52].

La version « groupes anonymes » risque donc bien de devenir une version historique ayant surtout permis la mise au point du programme.

4.2.1 Session de juin

Pour cette expérience, le paramètre de distance entre deux épreuves à été fixé à 2 (valeur lue dans le fichier *param.txt*), ce qui garantit que l'étudiant ne présente pas deux épreuves le même jour.

Labélisation

Dans cette version, il y a nécessité d'une labélisation des variables liées aux étudiants. Pour rappel, ces variables apparaissent dans des expressions du type:

passe(Nom, Cours, Type, X)

Le module général de labélisation est donc le suivant:

```
label_v2(FDE,FDP):-
    label_prof(FDP),
    label_etud(FDE).
```

La labélisation des variables liées aux professeurs se fait selon le principe suivant:

- priorité aux professeurs dont la disponibilité est la plus faible.

Pour les variables liées aux étudiants, le nombre d'épreuves à présenter étant sensiblement égal, aucun classement particulier n'est effectué.

---

[51] C'est l'expression qu'elle-même utilise.

[52] Cette vérification est pratiquement impossible lors d'une conception manuelle.

Heuristique

L'heuristique choisie pour les étudiants comme pour les professeurs est celle d'un choix aléatoire des variables.

Résultats

HORAIRE DE LA SESSION			
Jours	1ere M	2eme M	3eme M
31-mai AM		info2211:ecrit	
31-mai PM			info2327:oral(3)
1-juin AM			
1-juin PM			
2-juin AM	info2109:oral(1) info2112:oral(4)	info2226:ecrit	info2325:oral(1)
2-juin PM			
3-juin AM	info2102:oral(4)	info2201:ecrit	
3-juin PM	info2110:oral(1)		
4-juin AM	info2102:oral(3)	info2211:oral(1) info2204:ecrit	
4-juin PM		info2208:oral(4)	info2324:oral(1)
5-juin AM			
5-juin PM	ielv2113:oral(1)	info2232:ecrit	
6-juin AM			
6-juin PM	info2101:ecrit		info2304:oral(1)
7-juin AM		info2205:oral(1)	
7-juin PM	info2107:ecrit		
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2112:oral(2)	info2209:ecrit info2210:oral(1)	
10-juin PM	info2112:oral(3)		info2322:oral(4)
11-juin AM		info2223:oral(1) info2230:oral(2)	
11-juin PM	info2109:oral(4) info2102:oral(2)	info2230:oral(3)	info2302:oral(1)
12-juin AM		info2211:oral(4)	
12-juin PM		info2230:oral(1) info2208:oral(3)	info2301:ecrit
13-juin AM	info2105:ecrit		
13-juin PM		info2229:ecrit	ielv2307:oral(1)
14-juin AM	info2109:oral(3)		
14-juin PM		info2231:oral(1) info2221:oral(1)	info2327:oral(2)
15-juin AM			
15-juin PM			
16-juin AM	info2110:oral(2)	info2230:oral(5)	info2326:oral(1)
16-juin PM	info2112:oral(1)	info2208:oral(1)	
17-juin AM	info2114:oral(1)	info2211:oral(3)	

		info2230:oral(4) info2208:oral(2)	
17-juin PM			info2306:ecrit
18-juin AM	info2111:ecrit		
18-juin PM		info2208:oral(5)	info2324:oral(2)
19-juin AM	info2105:oral(1) info2111:oral(2)	info2233:oral(1)	info2322:oral(1)
19-juin PM	info2111:oral(3)		
20-juin AM	info2105:oral(2)		info2322:oral(3) info2301:oral(5)
20-juin PM	info2105:oral(3)	info2233:oral(5) info2211:oral(2)	
21-juin AM	info2105:oral(4)		
21-juin PM	info2105:oral(5)	info2233:oral(4)	info2301:oral(1) info2324:oral(3)
22-juin AM			
22-juin PM			
23-juin AM	info2106:ecrit	info2233:oral(3)	info2301:oral(2)
23-juin PM			
24-juin AM	info2109:oral(2) info2111:oral(1)	info2233:oral(2)	
24-juin PM			info2322:oral(2)
25-juin AM		info2233:oral(6) info2222:ecrit sent2288:ecrit	
25-juin PM			info2301:oral(4)
26-juin AM	info2108:ecrit	ielv2212:oral(1)	
26-juin PM	"		info2326:oral(2)
27-juin AM	info2110:oral(3)	info2202:ecrit	
27-juin PM	info2102:oral(1)	info2202:oral	info2327:oral(1) info2301:oral(3)

On constate que les écrits se répartissent davantage sur l'ensemble de la session avec une concentration plus importante au début de celle-ci. C'est attendu dans la mesure où les écrits ne sont plus explicitement placés en début de session[53]. Malgré tout, la contrainte qui exige que les écrits précèdent les oraux confine une majorité de ceux-ci dans la première moitié de la session.

Observez que le 4 juin, les épreuves des cours *info2211* et *info2204* (écrite) ont lieu en même temps. Cela ne pose aucun problème dans la mesure où ces deux épreuves n'ont pas d'étudiants communs. Une telle situation aurait été refusée dans la version « groupes anonymes ». On retrouve une situation analogue le 10 juin et le 25 juin.

Nous pouvons aussi nous intéresser aux horaires individuels des étudiants. Il est évidemment difficile de les examiner tous puisqu'ils concernent l'ensemble des étudiants des trois années de maîtrise (et licence). Nous en présentons quelques-uns en sachant qu'en première session, le nombre d'épreuves varie peu d'un étudiant à l'autre, ce qui n'est évidemment pas le cas en seconde session. De plus amples résultats sont présentés plus loin.

Voici l'horaire d'un étudiant de 2e maîtrise.

---

[53] Nous pourrions continuer à l'exiger mais nous ne l'avons pas fait par souci d'expérimenter d'autres choses.

achbany		
31-mai AM	info2211	ecrit
2-juin AM	info2226	ecrit
3-juin AM	info2201	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
12-juin AM	info2211	oral(4)
13-juin PM	info2229	ecrit
16-juin AM	info2230	oral(5)
18-juin PM	info2208	oral(5)
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

Il est normal que deux examens soient prévus le 27 juin puisqu'il s'agit d'un écrit et de son oral de rattrapage. Voici maintenant l'horaire d'un étudiant de 1ère maîtrise.

andre		
2-juin AM	info2112	oral(4)
3-juin AM	info2102	oral(4)
5-juin PM	ielv2113	oral(1)
6-juin PM	info2101	ecrit
7-juin PM	info2107	ecrit
11-juin PM	info2109	oral(4)
13-juin AM	info2105	ecrit
18-juin AM	info2111	ecrit
19-juin PM	info2111	oral(3)
21-juin PM	info2105	oral(5)
23-juin AM	info2106	ecrit
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit
27-juin AM	info2110	oral(3)

Le groupe d'oral auquel appartient l'étudiant est indiqué à titre d'information. L'identifiant du cours peut être avantageusement remplacé par la dénomination du cours mais nous avons préféré momentanément un tel affichage qui nous facilite les vérifications.

Enfin, voici l'horaire d'un étudiant de 3e maîtrise.

buyle		
31-mai PM	info2327	oral(3)
2-juin AM	info2325	oral(1)
10-juin PM	info2322	oral(4)
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)

buyle		
17-juin PM	info2306	ecrit
18-juin PM	info2303	ecrit
20-juin AM	info2301	oral(5)

Au niveau du langage, le choix d'une heuristique concernant les variables à choisir est également possible. Ainsi, en choisissant de fixer d'abord les variables ayant le moins de valeurs dans leur domaine, on obtient une solution assez équivalente à la précédente.

```
fd_labeling(X,[value_variable(first_fail), value_method(min)])
```

L'option *first_fail* est en quelque sorte l'équivalente d'un labeling classique avec tri préalable en fonction du nombre des valeurs qui sont encore disponibles.

achbany		
31-mai PM	info2205	oral(1)
2-juin AM	info2226	ecrit
3-juin AM	info2211	ecrit
5-juin AM	info2232	ecrit
6-juin AM	ielv2212	oral(1)
7-juin PM	info2210	oral(1)
10-juin AM	info2201	ecrit
13-juin AM	info2202	ecrit
13-juin PM	info2202	oral
16-juin AM	info2229	ecrit
21-juin AM	info2211	oral(4)
24-juin AM	info2230	oral(5)
25-juin AM	info2233	oral(6)
26-juin AM	info2208	oral(5)

4.3 Les échantillons considérés

4.3.1 Licences et maîtrises en informatique 2e session 2002

Les données de cette session d'examen nous ont permis d'établir une première version du programme *Prolog*. Nous avons, lors de cette première expérience, constitué les fichiers de données concernant les inscriptions des étudiants aux épreuves de manière artisanale sur base des listes sur lesquelles étaient imprimés les horaires. Cette première version a fourni des résultats qui paraissaient intéressants sur base de données que l'on savait fiables mais il est clair que la constitution d'un horaire de seconde session s'avère nettement plus simple parce que moins contraignant. Les difficultés éventuelles viennent des étudiants qui doivent présenter l'intégralité des épreuves[54].

Une version « groupes anonymes » et une version « horaires individuels » existaient déjà à ce moment.

En même temps, nous avons commencé à développer une interface web en constituant des données fictives pour les tests.

---

[54] ...et qui parfois se désistent peu de temps avant le début de la session pour la plus grande frustration du concepteur

Par la suite, lorsque nous avons pu disposer des données plus abondantes pour la session de juin 2003, ce programme a montré un certain nombre de lacunes, notamment en ce qui concerne sa robustesse. Il s'est avéré que les données exactes concernant les étudiants n'étaient parfois connues que fort tard, certains changements ayant pu avoir lieu en cours d'année et les mises à jour n'étant pas réalisées immédiatement. De ce fait, il n'a pas bien résisté au manque de consistance des données et nous avons décidé d'en réécrire une nouvelle version, dans laquelle les spécifications ont permis de mettre le doigt sur les problèmes potentiels. Dans le même temps, cette nouvelle version nous a permis d'améliorer la modélisation des données concernant les types d'épreuves, modélisation qui pourrait encore s'améliorer dans l'avenir. Nous l'évoquons dans le chapitre 6.

4.3.2 Licences et maîtrises en informatique 1ère session 2003

Nous n'avons pu disposer des données complètes pour pouvoir réaliser un horaire pour juin. En réalité, les données que nous avons utilisées contenaient des incohérences qui empêchaient le programme de fonctionner et le temps nous a manqué pour explorer soigneusement ces données afin de produire un horaire dans les temps[55].

La plus grosse incohérence rencontrée est celle de plusieurs étudiants inscrits plusieurs fois au même cours. La nature des contraintes est telle que l'impossibilité de trouver une solution ne peut être détectée avant que toutes les solutions aient été explorées ce qui conduisait le programme a fonctionner pendant un temps infini. Nous évoquons également ce problème dans les perspectives en proposant une solution plus locale pour la mise en jour de ces données.

Nous avons pu traiter les données de la 2e session avec la version réécrite du programme, pratiquement dans le même temps que son traitement manuel. Le traitement des données de la 1ère session 2003 a donc eu lieu **après** celui de la 2e.

Disposant en juillet des données (supposées) correctes de juin, nous avons pu faire tourner le programme et produire des résultats.

Si ces données ne contenaient plus d'incohérences, l'inscription des étudiants aux cours ne correspondait pas tout à fait à l'inscription aux examens. C'était notamment le cas des étudiants inscrits à titre principal dans une année et à titre complémentaire dans une autre. Il est clair que ces étudiants ne présentent pas, au cours de la même session, toutes les épreuves des deux années.

À titre d'exemple, parmi les données reçues, on pouvait remarquer qu'un de ces étudiants était inscrit à 22 cours et devait présenter 27 épreuves. Malgré tout, en ramenant à une seule unité le délai entre deux épreuves d'un même étudiant, le programme a fourni des solutions.

Après consultation de la secrétaire, il nous a été permis de corriger les données de la dizaine d'étudiants au statut particulier. De ce fait, nous avons pu allonger à 3 unités le délai entre deux épreuves.

4.3.3 Licences et maîtrises en informatique 2e session 2003

Ce sont les données de cette session qui ont, dans un premier temps, permis de tester l'efficacité du programme nouvelle version. Par rapport à l'horaire constitué manuellement, nous nous sommes donné les mêmes contraintes et nous en avons ajouté quelques-unes.

Outre les contraintes de disponibilité des professeurs, nous avons tenu compte des contraintes qui fixaient le jour de certaines épreuves coïncidant avec les mêmes épreuves en *LIHD*. Ce type de contraintes peut paraître anormal puisque l'ensemble des données devrait pouvoir être traité. Néanmoins, la constitution d'un horaire (fut-il un horaire d'examens) doit limiter son domaine à un moment donné et considérer les données extérieures à ce domaine comme des contraintes fortes.

---

[55] « Dans les temps » signifie en réalité fort tôt, avant le départ en congé de Pâques des étudiants.

Pour donner un exemple, supposons qu'un professeur enseigne à la faculté de Droit mais qu'il donne un seul cours à l'institut d'Informatique. Un programme d'horaire pourrait traiter l'ensemble des données, celles qui concernent la faculté de Droit et celles qui concernent l'institut d'Informatique. Au-delà de ça, on pourrait imaginer que le programme prenne en charge l'ensemble des examens des Facultés Universitaires Notre-Dame de la Paix[56]. Mais là encore, le domaine n'est pas assez vaste puisque certains professeurs peuvent enseigner ailleurs qu'aux *FUNDP*.

Bref, il faut bien s'arrêter quelque part et fixer des contraintes trop fortes pour limiter le domaine. Dans le cas présent, les épreuves de la *LIHD* étant fixées, celles-ci ont imposé, au moment du test, que certains examens se déroulent à des dates très précises. Il est clair qu'en s'y prenant suffisamment tôt, les données de la *LIHD* pourraient être incorporées à celles des licences et maîtrise pour la constitution d'un horaire commun.

Par rapport à l'horaire conçu manuellement, l'horaire produit automatiquement fournit évidemment la garantie que les étudiants ne doivent jamais présenter deux épreuves en même temps, même si ces derniers doivent présenter des épreuves correspondant à deux années d'étude. Le programme a pu fonctionner en enlevant les samedis d'examen à l'exception du premier et pour une question évidente de contrainte[57].

D'une manière générale, qu'il s'agisse de la première ou de la seconde session, quelques progrès sont encore à réaliser pour améliorer la qualité de ces horaires. Les possibilités de développement incrémental que nous évoquons dans le chapitre 6 et la souplesse possible au niveau de la modélisation des données nous donnent les pistes pour réaliser ces améliorations.

---

[56] Nous avons l'ambition de faire tourner ce programme sur l'ensemble des données des étudiants de la faculté des Sciences.

[57] Un examen oral devait avoir lieu impérativement le premier lundi de la session. Du coup, l'écrit qui, dans la version actuelle, précède toujours l'oral devait être placé ce samedi.

Chapitre 5 INTERFACE DE SAISIE DES DONNÉES

Entre ce qu'on cherche à exprimer, ce qu'on parvient à exprimer et ce que les gens comprennent, le mode de communication tient plus du téléphone arabe que de la photocopie. (Isabelle Alonso)

5.1 Les différentes sortes de données

Comme nous le verrons, certaines données sont très variables et demandent donc d'être puisées « à l'extérieur ». D'autres données le sont beaucoup moins et peuvent être gérées localement.

5.1.1 Les données concernant les étudiants

L'essentiel des données est constitué par l'ensemble des cours auxquels les étudiants sont inscrits et plus particulièrement, les cours dont ils doivent présenter les épreuves. Ces données, extrêmement variables d'une année à l'autre, sont théoriquement disponibles dans la base de données des étudiants. Il apparaît, pour des raisons qui sont expliquées ci-dessous, que ces données sont incomplètes. Dès lors, certaines difficultés sont à surmonter. Nous avons gracieusement obtenu ces données sous forme d'un fichier *Excel*. Nous avons copié ce tableau dans un document *Word* et transformé son contenu pour obtenir la structure désirée au moyen d'une macro-commande pour l'enregistrer sous forme d'un fichier texte dont voici un exemple de contenu:

```
[inscrit(achbany,[info2229,ielv2212,info2201,info2202,info2205,info2
208,info2210,info2211,info2226,info2230,info2232,info2233]),

inscrit(anciaux,[ielv2212,info2201,info2202,info2205,info2208,info22
10,info2211,info2221,info2224,info2230,info2232,info2233]),

inscrit(andre,[ielv2113,info2101,info2102,info2105,info2106,info2107
,info2108,info2109,info2110,info2111,info2112,sent2288]),

inscrit(arman,[info2229,ielv2212,info2201,info2202,info2205,info2208
,info2210,info2211,info2224,info2230,info2232,info2233]),

inscrit(barras,[ielv2212,info2201,info2202,info2205,info2208,info221
0,info2211,info2223,info2224,info2230,info2232,info2233]),

... ,

inscrit(zuyderhoff,[info2322,ielv2307,info2301,info2302,info2303,inf
o2306,info2325,info2327,info2328])

].
```

Ce fichier est directement utilisable par le programme *Prolog* puisque son contenu est identifiable à un terme qui sera lu en une seule fois pour devenir une liste[58]. Dans l'exemple qui suit, le fichier se nomme *epr83.txt* (données du mois d'août 2003) et la liste *LE*.

```
cree_liste('inscr83.txt',LE)
```

---

[58] Le langage que nous utilisons considère le point comme séparateur de termes. Il suffit donc que le fichier lu contienne la liste terminée par un point pour que celle-ci soit lue intégralement.

Voici le code qui détaille cette procédure:

```
cree_liste(File,L):-
    open(File,read,S),
    read(S,L),
    close(S).
```

La variable *File* désigne le fichier à prendre en compte et la variable *L*, la liste à créer. Quant à la variable *S*, elle désigne le flux de données (*stream*) à prendre en compte lors de la lecture, en l'occurrence ici, celui constitué par le fichier ouvert.

Dans l'espoir d'une consistance possible des données au niveau de la base de données des étudiants, on peut considérer que la simple mise à disponibilité du fichier *Excel* permet, en deux étapes simples, de préparer le fichier directement utilisable par le programme.

5.1.2 Les données concernant les cours

Ces données peuvent être considérées comme relativement peu variables. Elles le sont tout malgré tout. Elles concernent la liste des cours (identifiant, dénomination, types d'épreuves). Ces données sont disponibles pour le programme sous forme d'une liste à travers un fichier texte au moyen d'une instruction semblable à:

```
cree_liste('epr83.txt',LC)
```

Voici un exemple de contenu d'un tel fichier. Chaque cours possède son identifiant officiel et un type d'épreuve associé. La signification de chacun des types possibles a été détaillée dans le chapitre 2 consacré à la modélisation.

```
[  epreuve(info2211,oral(12)),
   epreuve(info2233,oral(9)),
   epreuve(info2210,ecrit+oral(999)),
   epreuve(info2110,oral(12)),
   epreuve(info2208,oral(12)),
   epreuve(info2232,ecrit),
   ...
   epreuve(info2202,ecrit*oral),
   epreuve(info2230,oral(6)) ].
```

C'est le type d'épreuves, donc la deuxième composante de chacun de ces éléments, qui est le plus sujet à des changements. Ces changements peuvent se produire d'une session à l'autre. Un professeur peut souhaiter interroger par écrit en première session et oralement en seconde, par exemple. Ils peuvent également se produire dès que le professeur décide de changer son mode d'interrogation. Il importe donc de donner à un utilisateur potentiel (la secrétaire, par exemple[59]), la possibilité d'effectuer des modifications ou des ajouts dans une petite base de données locale.

Une interface de type web, décrite dans une des sections qui suivent, permet à cet utilisateur potentiel de mettre à jour cette base de données locale créée avec *MySQL*. La création des pages est dynamique (*PHP*) et la mise à jour est autorisée à travers le remplissage des formulaires qu'elles contiennent.

---

[59] Dans un premier temps, nous avions pensé donner à chaque enseignant la possibilité d'encoder lui-même ses données, mais il nous a paru plus pertinent, pour des questions d'efficacité, de confier à une seule personne le soin de rassembler les données et de contrôler les délais.

L'utilisation de cette interface permet donc la modification et, antérieurement, la création d'autres fichiers nécessaires au programme *Prolog*.

5.1.3 Les données concernant les professeurs

Ces données sont également peu variables, les attributions des professeurs restant relativement stables. Elles concernent d'abord les cours enseignés[60] par le professeur (identifiant du professeur, liste de cours). Ces données sont disponibles pour le programme sous forme d'une liste à travers un fichier texte au moyen d'une instruction semblable à:

```
cree_liste('profs83.txt',LP)
```

Voici un exemple de contenu d'un tel fichier.

```
[  enseigne( ade, [ info2114 , info2328 ]),
   enseigne( ble, [ info2109 ]),
   enseigne( clo, [ info2102, info2304 ]),
   ...
   enseigne( pre, [ info2111 ]),
   enseigne( pys, [ info2108, info2326, info3101 ]),
   enseigne( ven, [ info2208, info2232, info2305, info2325 ])].
```

L'identifiant du professeur est classique et se compose de trois lettres correspondant généralement à la première lettre de son prénom suivie des deux premières lettres de son nom. La règle souffre cependant quelques exceptions. L'interface web doit permettre d'ajouter ou de retirer des professeurs et des cours dans la base de données *MySQL*.

Les données liées au professeur concernent également leur indisponibilité en cours de session dès lors que leur présence est considérée comme nécessaire lors des épreuves qu'ils font passer. Ces données doivent évidemment être collectées lors de chaque session. Ces données sont disponibles pour le programme sous forme d'une liste à travers un fichier texte au moyen d'une instruction semblable à:

```
cree_liste('indis83.txt',LI)
```

Voici un exemple de contenu d'un tel fichier.

```
[
indispo(jbe,[1,2,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
36,37,38]),
indispo(fbo,  [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32]),
indispo(obo, []),
indispo(jnc, [10,11,12,13,14,15,16,17,18,19,20,21,22]),
...
indispo(jva2, [1,2]),
indispo(cvw, [])].
```

---

[60] Inutile de préciser que la notion de cours "enseigné" est directement liée aux épreuves à faire passer. Par exemple, si un professeur a l'habitude de faire surveiller un écrit par son assistant, c'est à l'assistant qu'il faut attribuer le cours dans le contexte qui nous préoccupe.

On retrouve à nouveau l'identifiant du professeur et la liste numéros des demi-jours d'indisponibilité. Concrètement, la constitution de ces listes ne pose guère de problèmes. L'utilisateur de l'interface web sélectionne pour chaque professeur l'ensemble des demi-jours d'indisponibilité dans une liste déroulante où la sélection multiple est possible et la programmation s'occupe de générer les bons fichiers dans la base de données.

5.1.4 Les autres données

Les **dates des demi-jours session** font partie des données nécessaires à l'affichage de l'horaire (sinon, le nombre de demi-jours suffirait). Ces données sont disponibles pour le programme sous forme d'une liste à travers un fichier texte au moyen d'une instruction semblable à:

```
cree_liste('dates83.txt',LD)
```

Voici un exemple de contenu d'un tel fichier.

```
[  date( 1 ,' 16-août AM '),
   date( 2 ,' 16-août PM '),
   date( 3 ,' 17-août AM '),
   ...
   date( 35 ,' 02-sept AM '),
   date( 36 ,' 02-sept PM ')].
```

À chaque numéro de demi-jour qui est identifiant, est associé le « texte » de la date (jour, mois, avant ou après-midi). Dans l'état actuel du développement, ce fichier est édité manuellement. Pour la facilité, il n'est pas tenu compte des jours où il n'est pas possible d'interroger (singulièrement les dimanches et jours fériés). Un autre fichier contient cette information.

Il est utile de placer certains **paramètres** dans un fichier de configuration. Ces données sont disponibles pour le programme sous forme de variables à travers un fichier texte au moyen d'instructions semblables à:

```
cree_liste('param83.txt',[NDJ,NGMax,_])
```

ou

```
cree_liste('param83.txt',[_,_,D])
```

Actuellement, trois paramètres peuvent être fixés manuellement:

- le nombre de demi-jours de la session,
- le nombre de groupes maximum à constituer pour un oral,
- le nombre de demi-jours souhaités entre deux épreuves d'examens d'un même étudiant.

Les deux premiers paramètres sont surtout utiles pour limiter dès le départ le domaine des variables de domaine fini correspondant aux dates de passage des étudiants (et fatalement, d'interrogation des professeurs). Le troisième se ramène au minimum dans le cas d'une seconde session si au moins un des étudiants doit présenter toutes les épreuves (session complète).

Il faut également préciser les dates des (demi-)jours où il est impossible d'interroger (les dimanches, le lundi de Pentecôte, éventuellement certains samedis). Nous les avons qualifiés de (demi-)jours de **congé**. Ces données sont disponibles pour le programme sous forme d'une liste à travers un fichier texte au moyen d'instructions semblables à:

```
cree_liste('conges83.txt',LCong)
```

Voici un exemple de contenu d'un tel fichier.

```
[3,4,15,16,17,18,29,30,31,32].
```

Ce fichier est actuellement édité manuellement. Il est clair que la génération de ce fichier est possible via l'interface web.

5.2 Les étudiants au statut particulier

Certains étudiants sont inscrits dans une année à titre principal et dans une autre à titre complémentaire. De ce fait, il est quasi sûr qu'ils ont déjà présenté un certain nombre d'examens par le passé et qu'ils bénéficient de reports de cotes pour ceux-ci. Dans la base de données des étudiants, ces informations ne sont pas enregistrées. En réalité, il semble qu'elles n'existent pas sous forme numérique. Les étudiants qui sont dans ce cas sont donc inscrits à un nombre important de cours (plus d'une vingtaine). Un horaire qui tient compte de tous les cours auxquels les étudiants sont inscrits prendra en compte des contraintes beaucoup trop importantes[61].

5.3 La disponibilité des données

Il importe donc, pour faire fonctionner correctement le programme, de pouvoir disposer des informations significatives à savoir, les épreuves que les étudiants présenteront réellement. Ces informations nous ont été fournies sous forme électronique pour la seconde session de l'année académique 2002-2003. Pour la première session, elles ont été mises à jour tardivement. Notre programme a donc fonctionné sur des données incomplètes. Lors d'un essai ultérieur à la session, les données n'étaient plus disponibles qu'en terme d'inscription aux cours. Nous avons donc effectué manuellement les corrections pour les étudiants au statut spécial, tenant compte des listes disponibles dans les horaires établis par le secrétariat, ceci ne nous garantissant pas l'exactitude des données restantes. Toutefois, si des différences devaient exister, elles iraient certainement dans le sens d'un trop grand nombre d'épreuves prévues, ce qui est plutôt rassurant quand au fonctionnement du programme.

5.4 L'interface web

Nous ne présentons ici que des parties des écrans les plus significatifs pour la compréhension du fonctionnement de l'application. Pour visualiser l'ensemble des écrans complets, nous vous renvoyons aux annexes.

5.4.1 Se connecter à l'application

L'utilisateur est invité à se connecter en fournissant identificateur et mot de passe. S'il n'est pas connu du système, il peut s'inscrire. S'il décide de s'inscrire, il doit fournir un minimum de renseignements au système tels son nom, son prénom, le choix d'un identificateur en 8 caractères maximum et d'un mot de passe de 6 caractères au minimum et reproduction de celui-ci pour confirmation.

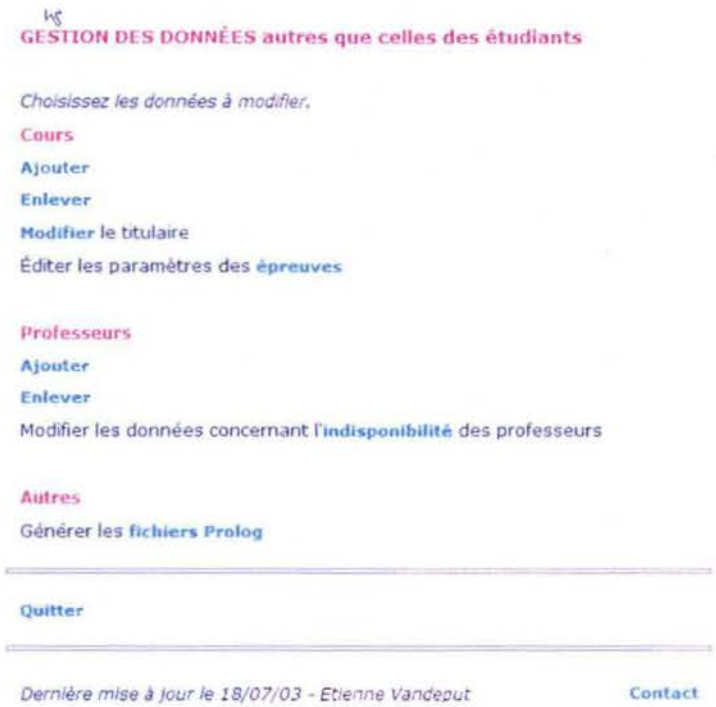
Les différents problèmes accompagnant la fourniture des informations sont gérés soit au niveau du client (en *Javascript*) soit au niveau du serveur (en *PHP*, par la production d'une page indiquant l'erreur). Plusieurs exemples sont illustrés en annexe.

---

[61] Dans le chapitre consacré aux résultats, nous avons malgré tout relaté l'existence de solutions pour de tels problèmes sur-contraints. Dans l'exemple donné, un étudiant devait présenter 27 épreuves! Il est clair qu'un horaire constitué sur de telles bases ne peut fournir des solutions de qualité.

5.4.2 Gérer les informations

Dans les deux cas, et lorsque les erreurs sont corrigées, il peut alors commencer à gérer la base de données locale et voit s'afficher des rubriques concernant les cours, les professeurs et la génération des



fichiers *Prolog* (un aspect plus global de la page est visible en annexe).

5.4.3 Gérer les informations concernant les cours

Dans l'état actuel de développement, et pour des raisons qui sont expliquées par ailleurs, les informations concernant les étudiants ne sont pas gérées via cette interface. L'utilisateur peut ajouter un cours, en supprimer un, en modifier le titulaire ou encore modifier les desiderata en matière d'épreuves pour ce cours (oral, écrit, le même jour,...).

Nous présentons ici une partie de l'écran de modification des desiderata en matière d'épreuves pour un cours accessible par le lien de la rubrique *Éditer les paramètres des épreuves*. Les modifications s'effectuent pour l'ensemble des cours d'un même professeur. Les choix actuellement possibles portent sur le type d'épreuve (écrite et/ou orale) sur le nombre d'étudiants à interroger par demi-jour (la valeur 999 est choisie pour les professeurs qui ne souhaitent qu'un seul groupe pour l'oral). Certains enseignants souhaitent également que l'épreuve écrite d'un cours soit placée le matin et suivie l'après-midi, d'un oral de rattrapage pour l'ensemble des étudiants qui le souhaitent, ce que permet l'option *Même jour que l'écrit*.

Lorsque tous les choix sont effectués, ils sont validés par l'utilisateur et un écran récapitulatif est affiché. Un exemple de ce type d'écran est également proposé.

EXIGENCES EN MATIÈRE D'EXAMENS

COURS ihdc2004

Ecrit ☐

Oral ☑

Nbre d'étudiants par demi-jour [10] *(999 si un seul groupe à constituer)*

Même jour que l'écrit ☐

COURS info2222

Ecrit ☑

Oral ☑

Nbre d'étudiants par demi-jour [999] *(999 si un seul groupe à constituer)*

Même jour que l'écrit ☑

COURS info2004

Ecrit ☑

Oral ☐

Nbre d'étudiants par demi-jour [0] *(999 si un seul groupe à constituer)*

Même jour que l'écrit ☐

Envoyer

Quitter

Dernière mise à jour le 21/07/03 - Etienne Vandeput **Contact**

DESIDERATA EN MATIÈRE D'EXAMENS

Voici la liste des épreuves que souhaite faire passer le professeur dont l'identifiant est jpl:

COURS ihdc2004

épreuve orale: 10 étudiants par demi-jour

COURS info2222

épreuve écrite

épreuve orale

épreuves écrite et orale le même jour

COURS info2004

épreuve écrite

Retour à la page de modification des **paramètres des cours**

Retour à la page de **gestion**

Dernière mise à jour le 22/07/03 - Etienne Vandeput **Contact**

Il est à remarquer que les épreuves de certains cours pourraient ne pas être fixées, auquel cas ces cours ne sont pas pris en considération par le programme.

5.4.4 Gérer les informations concernant les professeurs

L'utilisateur peut ajouter et retrancher un professeur de la base de données. Il est intéressant de noter que **lorsqu'un professeur est retranché de la base de données, les cours qui lui étaient attribués ne le sont pas**, ce qui permet de les attribuer à un autre professeur via la modification de titulaire décrite au paragraphe précédent.

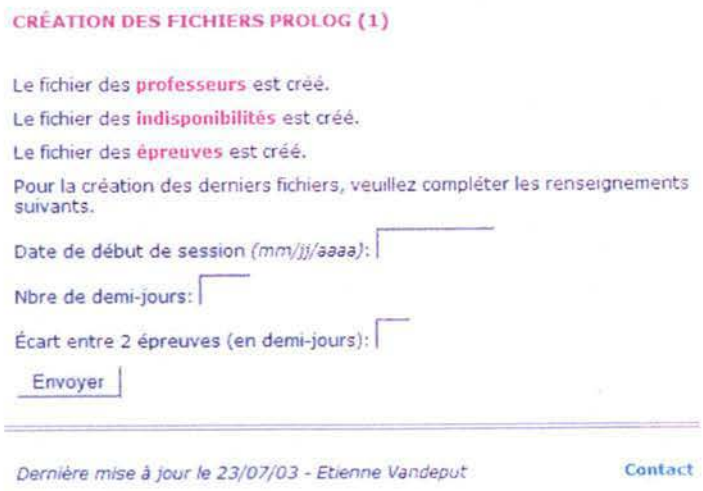
L'option la plus intéressante est celle qui permet de fixer les jours d'indisponibilité de chaque professeur.

La liste des demi-jours est générée automatiquement sur base des renseignements fournis par l'utilisateur concernant le début de la session et le nombre de demi-jours de celle-ci, ce qu'il réalise en suivant le lien correspondant à la fonctionnalité qui est décrite ci-après.

En cas de modification de l'indisponibilité, les données préalablement enregistrées sont évidemment récupérées.

5.4.5 Générer les fichiers nécessaires au programme Prolog

Pour rappel, le programme *Prolog* a besoin pour fonctionner de fichiers textes contenant les informations sous forme de liste d'éléments composés. Dans un premier temps, nous avons utilisé la possibilité de transformer les tables de la base de données en fichiers textes grâce au programme *mysqldump*. Récupérés avec un programme de traitement de texte, une macro-commande permettait de donner au texte le format nécessaire au programme. De manière plus pratique, cette transformation se fait maintenant directement par l'intermédiaire de *PHP*, ce qui permet à l'utilisateur de demander directement la génération de ces fichiers.



Voici un bout de code commenté qui illustre comment cette transformation est possible.

```
<?php
```

Connexion à *MySQL*, à la base de données et sélection de tous les enregistrements de la table *professeurs*

```
$connexion=mysql_connect("localhost","root","")
 or die ("<p>La connexion avec le serveur ne peut être établie.</p>");
$select=mysql_select_db("Examens")
 or die ("<p>La base de données 'Examens' ne peut être sélectionnée.</p>");
$sqlquery="SELECT * FROM professeurs";
$queryresult=mysql_query($sqlquery)
 or die ("<p>La requête a échoué.</p>");
```

Nombre d'enregistrements trouvés

```
$nl=mysql_num_rows($queryresult);
```

Initialisation des chaînes à obtenir (représentant des listes): données concernant les indisponibilités des professeurs (fichier *indis.txt*) et données concernant les types d'épreuves (fichier *epr.txt*)

```
$ch="[\n";
$ch2="[\n";
```

Pour chaque professeur…

```
for($i=0;$i<$nl;$i++){
```

…considérer l'enregistrement courant…

```
  $r=mysql_fetch_row($queryresult);
```

…ajouter l'expression *indispo(nom_du_prof,[liste_des_jours])* à la chaîne représentant la liste des indisponibilités,

```
  $ch=$ch."indispo(".$r[0].",[".$r[1]."]),\n";
```

Sélectionner les cours de ce professeur…

```
$sqlquery2="SELECT * FROM cours WHERE user='".$r[0]."'";
$queryresult2=mysql_query($sqlquery2)
 or die ("<p>La requête a échoué.</p>");
$nl2=mysql_num_rows($queryresult2);
```

…ajouter à la chaîne représentant la liste des cours enseignés, l'expression *enseigne(nom_du_prof,[*

```
$ch2=$ch2."enseigne(".$r[0].",[";
```

Pour chaque cours de ce professeur…

```
for($j=0;$j<$nl2;$j++){
  $r2=mysql_fetch_row($queryresult2);
```

…ajouter à la chaîne représentant la liste des cours enseignés, l'expression *nom_du_cours,*

```
  $ch2=$ch2.$r2[0].",";
}
```

…enlever la dernière virgule si nécessaire (c-à-d. lorsqu'il y a des éléments dans la liste)…

```
if ($nl2!=0){$ch2=substr($ch2,0,strlen($ch2)-1);}
```

…et compléter l'élément de la liste par *]),*

```
$ch2=$ch2."]),\n";
}
```

…enlever la dernière virgule de chacune des deux listes et les clôturer par *].*

```
$ch=substr($ch,0,strlen($ch)-2);
$ch2=substr($ch2,0,strlen($ch2)-2);
$ch.="\n]";
$ch2.="\n]";
```

Enregistrer les deux chaînes dans les deux fichiers

```
$f=fopen("indisp.txt","w");
$f2=fopen("profs.txt","w");
if (!$f || !$f2){
   echo "<p>Un des fichiers au moins n'a pu être ouvert.</p>";
}
else{
   fputs($f,$ch);
   fputs($f2,$ch2);
   fclose($f);
   fclose($f2);
```

Donner un feed-back sur l'opération

```
    echo "<p>Le fichier des <span class='important1'>professeurs</span> est
    créé.</p>";
    echo "<p>Le fichier des <span class='important1'>indisponibilités</span>
    est créé.</p>";
}
?>
```

Voici à quoi ressemble le contenu du premier de ces fichiers tel qu'il est traité par le programme:

```
[
indispo(fbo,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2
2]),
indispo(jbe,[9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,29,30,31
,32]),
indispo(jlh,[1,2,3,4,7,8,13,14,15,16,26,27,28]),
indispo(jmj,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,25,26,27,28,29,3
0]),
indispo(jpl,[3,4,5,6,15,16,23,24,25,26,31,32]),
indispo(jva,[7,8,9,10,11,12,13,14,15,16,23,24,25,26,27,28,31,32]),
indispo(mno,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,17,18,23,24,25,26,31,3
2]),
indispo(nha,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,29,3
0]),
indispo(obo,[1,2,3,4,15,16]),
indispo(rle,[]),
indispo(sve,[]),
indispo(ven,[1,2,3,4,5,6]),
indispo(yde,[9,10])
].
```

Le programme ouvrira ce fichier et en affectera le contenu à une liste *Prolog* directement utilisable. L'instruction qui le permet est la suivante:

```
cree_liste('indis.txt',LI)
```

avec comme description de la procédure *cree_liste*:

```
cree_liste(File,L):-
      open(File,read,S),
      read(S,L),
      close(S).
```

où *S* désigne un flux logique de données, la procédure *read* lisant un terme. Syntaxiquement parlant, c'est le point qui marque la fin du terme. Il n'y a donc qu'un seul terme à lire dans le cas présent.

Sont donc générés les fichiers concernant l'indisponibilité des professeurs, le fichier des types d'épreuves et le fichier des attributions de cours. Le fichier des dates est généré par la suite, lorsque l'utilisateur a fourni les derniers renseignements nécessaires.

Chapitre 6 PERSPECTIVES

L'avenir, c'est ce qui dépasse la main tendue.
(Louis Aragon)

6.1 Au niveau du développement de l'interface

6.1.1 Amélioration de l'interface existante

L'interface actuelle a été présentée et semble donner satisfaction. Toutefois, ce n'est qu'à l'usage que des imperfections apparaîtront, tant au niveau de l'utilité que de l'utilisabilité.

L'amélioration la plus importante devrait concerner la gestion individuelle des programmes des étudiants. Nous avons signalé que la difficulté majeure consiste à disposer d'un maximum d'informations au moment de la génération de l'horaire en ce sens que cette génération a lieu plusieurs semaines avant le début de la session. Un certain nombre d'informations nouvelles peuvent encore apparaître pendant ce délai concernant:

- les types d'épreuves ou les desiderata en matière d'examens,
- la disponibilité des professeurs (de nouvelles contraintes peuvent apparaître ou disparaître),
- le programme des étudiants (abandon avant le début de la session, reports de cotes,...),
- ...

Dans l'hypothèse où l'horaire généré ne peut être refaçonné que par génération d'un nouvel horaire tenant compte des nouvelles données (même si peu d'entre elles ont changé), il est souhaitable que ces données soient mises à jour le plus rapidement possible.

Actuellement, les données sont fournies par la gestionnaire de la base de données des étudiants qui fournit un fichier *Excel* reprenant l'ensemble des cours auxquels les étudiants sont **officiellement** inscrits. Des étudiants ayant des inscriptions sur plusieurs années sont donc officiellement inscrits à de nombreux cours dont, en réalité, ils ne passent pas toutes les épreuves, loin s'en faut. Il est donc nécessaire de pouvoir gérer **localement** ces données afin d'en avoir une mise à jour fidèle à la réalité.

La solution passe par la réception, le plus tôt possible dans l'année académique[62], de ce fichier *Excel* et de sa transformation en table *MySQL* que l'interface web existante et aménagée pourra gérer.

L'amélioration consistera donc en une série d'écrans permettant à l'utilisateur expert de modifier les programmes des étudiants avant de générer à nouveau (toujours via l'interface web) le fichier nécessaire au programme *Prolog*.

6.1.2 Développement de nouveaux modules

Comme nous l'expliquons dans la section suivante, une approche incluant la notion de « difficulté » d'une épreuve de même que l'acceptation partielle d'un résultat produit est envisageable.

Le développement d'une telle approche doit être accompagné de la modification des écrans concernant la saisie des paramètres des épreuves, ce qui constitue une modification mineure et de la création d'écrans permettant le choix des épreuves à « replacer ». Cette dernière s'avère nécessaire dans le cas d'une version incluant la possibilité de bloquer la position d'une majorité d'entre elles.

---

[62] ...en principe, à la date de clôture des inscriptions

6.2 Au niveau de l'approche de programmation

6.2.1 Amélioration de la version existante

Certaines améliorations n'ont été envisageables qu'à partir de l'examen des résultats produits. Citons principalement au rayon des constats les éléments suivants:

- certains examens réputés « plus difficiles » sont proches l'un de l'autre,
- il peut se produire qu'un professeur fasse passer un examen écrit d'un certain cours le matin et un oral d'un autre cours l'après-midi[63],
- dans les solutions fournies, on trouve une bonne proportion d'épreuves « bien placées ».

Les réponses à ces constats sont de plusieurs ordres.

Il faut d'abord signaler que les horaires examinés par l'utilisatrice experte sont ceux issus d'une première solution fournie par le programme et que celui-ci est capable d'en fournir bien d'autres[64]. La production de solutions supplémentaires, surtout s'il est fait usage d'heuristiques basées sur un choix aléatoire d'une partie des valeurs[65], donne des résultats assez différents de l'un à l'autre demandant une relecture quasi complète de l'horaire produit.

Un coefficient de difficulté

Pour ce qui est des deux premiers constats, les solutions produites peuvent être améliorées par une modélisation plus fine des cours. Dans l'état actuel du développement, les paramètres liés aux cours concernent les types d'épreuves à faire passer. Il est possible d'envisager un paramètre qui soit un coefficient de difficulté de l'épreuve à passer. Si l'on se trouve ici dans le domaine du subjectif, il n'en reste pas moins vrai que cet élément est pris en compte par les concepteurs d'horaires d'une manière, il est vrai, peu formalisée.

L'idée est d'associer à ces coefficients des contraintes plus ou moins fortes. Par exemple, une épreuve de coefficient 1 pourrait avoir lieu le même jour qu'une autre épreuve de coefficient 1. Une épreuve de coefficient 2 demanderait de ne pas être organisée le même jour qu'une autre épreuve et une épreuve de coefficient 3 se déroulerait au moins 4 périodes plus tard qu'une autre épreuve. Un système encore plus fin pourrait être mis en place. Parmi les éléments subjectifs pris en compte, on pourrait trouver, par exemple, la nécessité de placer l'épreuve le matin ou encore la charge de travail de correction.

```
epreuve(info2106,ecrit,3)
```

Cette amélioration devrait permettre de nuancer la contrainte d'espacement entre les épreuves qui est trop générale. Dans le cas d'une seconde session où certains étudiants doivent représenter toutes les épreuves, cette contrainte doit se limiter à un seul demi-jour, permettant ainsi que deux examens aient lieu le même jour. Ce n'est pas gênant pour certaines épreuves, ça l'est beaucoup pour d'autres.

Pour l'étudiant dont l'horaire individuel de seconde session est le suivant, la journée du 20 août risque d'être difficile car il a deux oraux à présenter. Pour la journée du 26 août, il s'agit du même cours et l'oral doit suivre l'écrit. Ce n'est donc pas un problème.

---

[63] Il semble que cette situation ne soit pas un critère favorisant la constitution d'un « bon » horaire.

[64] Seul un utilisateur expert est capable de juger finement de la qualité d'un horaire. De ce point de vue, il n'est pas décent de demander un examen attentif de nombreuses séries d'horaires à une personne qui réalise ce travail bénévolement. L'examen s'est donc souvent limité à une première solution produite. Notons que, même si un des buts de ce travail est de fournir une implantation correcte et efficace de l'application, nous ne sommes pas tout à fait dans les conditions d'un projet de développement commandé.

[65] ...garantissant notamment une bonne répartition des épreuves sur l'ensemble de la session.

16-août AM	info2112	oral(1)
18-août AM	info2108	ecrit
19-août PM	info2102	oral(1)
20-août AM	info2110	oral(1)
20-août PM	info2109	oral(2)
22-août AM	info2106	ecrit
26-août AM	ielv2113	ecrit
26-août PM	ielv2113	oral
27-août AM	info2107	ecrit
28-août PM	info2111	ecrit
29-août PM	info2101	ecrit
01-sept AM	sent2288	ecrit
02-sept AM	info2111	oral(1)

Au niveau du programme, les contraintes concernant le placement des épreuves seraient prises en compte en fonction des coefficients respectifs des cours concernés. La modélisation des cours est quelque peu modifiée et demande une refonte du programme qui, sans être énorme, demande tout de même une réécriture de certaines de ses parties. Les 23 et 24 sont un samedi et un dimanche qui, dans ce cas, ont été exclus des jours d'examens possibles. On peut constater que le lundi n'est pas utilisé et que le mardi est occupé par un examen de langues. Une solution avec coefficients pourrait améliorer une telle solution au profit de l'étudiant.

Un coefficient de charge de correction

L'expérience des concepteurs fait aussi mention du placement de certains cours en début de session, tenant compte ainsi d'une charge de correction élevée pour certains professeurs. Un autre coefficient pourrait s'ajouter, permettant ainsi au labeling de placer prioritairement ces cours en début de session.

```
epreuve(info2106,ecrit,3,1)
```

Un blocage de l'horaire pour une majorité d'épreuves

Les horaires produits s'avérant acceptables pour une majorité des épreuves et pour une majorité des acteurs concernés (professeurs et étudiants), on peut se demander s'il n'est pas préférable, plutôt que de rechercher d'autres solutions plus acceptables, de bloquer l'horaire pour un ensemble d'épreuves considérées comme « bien placées » et de demander au programme de recalculer d'autres périodes pour les épreuves « moins bien placées ». Il semble que ce soit une évolution inévitable si on ne veut pas se contenter d'une solution qui soit un squelette d'horaire complété par quelques changements manuels[66].

Une telle amélioration demanderait l'écriture d'un module qui tiendrait compte des indications fournies par l'utilisateur pour considérer comme contraintes de ne plus modifier les valeurs attribuées à un

---

[66] Comprenons-nous bien, les solutions produites par le programme dans sa version actuelle sont correctes en ce qu'elles tiennent compte des contraintes définies. Les améliorations manuelles dont il est question se rapportent à l'établissement d'un horaire de meilleure qualité tenant compte de critères qui n'ont pas été formalisés.

ensemble important d'épreuves. Le même programme fonctionnerait alors avec un ensemble de contraintes ajoutées.

Il est actuellement difficile d'évaluer l'ampleur de la tâche car elle concerne davantage le développement de l'interface que le programme de détermination de l'horaire lui-même.

6.2.2 *Autre point de vue théorique*

Dans un autre registre, on peut se demander si d'autres approches théoriques ne peuvent fournir de meilleures solutions. On se rappelle que les contraintes dites douces non pas été prises en compte dans notre approche. Ce qui peut être considéré comme un frein à la prise en compte d'un trop grand nombre de contraintes, et en tous cas, de contraintes qui ne doivent pas absolument être satisfaites. De ce point de vue, l'approche de la **programmation hiérarchique** nous semble intéressante bien que probablement très sophistiquée dans le contexte qui nous occupe. Nous en développons les principaux aspects dans le paragraphe qui suit.

6.3 *La programmation hiérarchique*

6.3.1 *Motivation*

Les contraintes dont il a été question jusqu'ici sont des contraintes qui doivent être absolument respectées par les solutions potentielles du problème. Or l'expression d'une contrainte peut assez souvent être accompagnée d'un commentaire adoucissant.

La contrainte: *« **On exige** une journée d'intervalle entre deux examens consécutifs d'un même étudiant. »* peut être avantageusement remplacée par la contrainte *« **Autant que possible, on souhaite** une journée d'intervalle entre deux examens consécutifs d'un même étudiant. »*

Nous avons là un bel exemple de la nuance importante qui existe entre la contrainte dure et la contrainte douce. Les contraintes dures ont tendance à éliminer des solutions qui pourraient être acceptables et à restreindre beaucoup trop vite et beaucoup trop fort les domaines des variables concernées. Elles peuvent être perçues, de ce point de vue, comme un obstacle à la production de solutions de bonne qualité. L'horaire produit respecte les contraintes imposées mais de nouvelles contraintes devraient être définies pour améliorer les solutions produites au risque de conduire à des blocages.

Si nous demandons que trois demi-jours au moins séparent deux épreuves consécutives d'un même étudiant, nous le demandons pour tous les étudiants, y compris ceux qui ont énormément d'épreuves à passer[67]. Il est intéressant de permettre au programme de prendre des libertés par rapport à des contraintes qui apparaissent trop restrictives dans certaines circonstances.

Cette problématique n'est pas propre à la constitution des horaires mais concerne une très large gamme de domaines allant de la gestion d'environnements de dessin assisté jusqu'au formatage de documents, en passant par la simulation en physique et bien d'autres domaines encore. Pour dire les choses autrement, nombreuses sont les applications où l'utilisateur est amené à fournir des paramètres à une application, et il ne le fait pas toujours dans le respect de toutes les contraintes qu'il a lui-même déjà fournies au programme. Pour éviter un blocage fatal à la progression de cet utilisateur pas toujours très rigoureux, le programme doit agir **au mieux**[68].

Exemple

Lors de la conception d'une page web, le concepteur demande la création d'un tableau en trois colonnes de 600 pixels de large. Il exige que la première colonne soit large de 300 pixels. Par la suite, il tente d'élargir dynamiquement les deux autres colonnes pour leur donner une largeur d'environ 250

---

[67] Nous l'avons déjà mentionné, ce problème est particulièrement perceptible dans le cas de la seconde session.

[68] Voilà encore une notion dont la signification, hélas, peut prendre différentes colorations.

pixels. Le logiciel qui génère le code et qui fournit une prévisualisation de la page doit traiter ces informations d'une manière ou d'une autre en agissant d'une des façons suivantes:

- élargir le tableau;
- rétrécir la première colonne;
- empêcher un élargissement dynamique trop important des colonnes de droite;
- mélanger les solutions précédentes
- ...

Cette illustration pourtant éloignée de la constitution des horaires montre bien l'intérêt des contraintes douces. Tout ce qui est demandé ne peut être satisfait, mais une solution est toutefois fournie.

Dans les lignes qui suivent, nous développons le schème **HCLP**[69] proposé par Molly Wilson et Alan Borning [1]. *HCLP* est une évolution de *CLP* qui permet une hiérarchisation des contraintes à plusieurs niveaux de préférence. *HCLP* s'appuie sur la technique *CLP*. Comme ce dernier, il est paramétré par un domaine mais il est également paramétré par ce qu'il est convenu d'appeler un comparateur.

6.3.2 *Hiérarchie de contraintes*

Voici quelques définitions et notations qui nous permettront de formaliser les choses.

Une contrainte est une relation sur un domaine (celui des variables FD pour le problème qui nous occupe)

- **Domaine**: $\mathcal{D}$

Les contraintes sont exprimées en utilisant des prédicats.

- Symboles des prédicats de contraintes: $\Pi_{\mathcal{D}}$
- **Contrainte**: $p(t_1, ..., t_n)$
  - p est un symbole n-aire de $\Pi_{\mathcal{D}}$
  - $t_1, ..., t_n$ sont des termes

En *HCLP*, les contraintes sont étiquetées par une « force », un « niveau d'exigence ». Ces niveaux sont totalement ordonnés. Une **hiérarchie de contraintes** est un ensemble fini de contraintes étiquetées.

- **Contrainte étiquetée**: lc (l est un niveau d'exigence)
- $H_0, ..., H_n$: vecteurs de contraintes ($k > n \Rightarrow H_k = \varnothing$)
  - H_0: contraintes requises
  - H_1: le niveau non requis de plus haute exigence
  - H_2: le niveau suivant
  - etc.

Une **valuation** (pour un ensemble de contraintes) est un jeu de valeurs de $\mathcal{D}$ pour chacune des variables libres de l'ensemble des contraintes.

Une **solution** (pour une hiérarchie de contraintes) est un ensemble de valuations pour les variables libres de la hiérarchie qui satisfont les contraintes non requises[70] **au moins aussi bien** que les autres valuations.

---

[69] Hierarchical Constraint Logic Programming

[70] Les contraintes requises doivent toujours être satisfaites.

Le terme « au moins aussi bien » doit être spécifié. Il nous conduit tout droit vers une autre démarche, celle du calcul des erreurs.

6.3.3 *Fonctions d'erreurs*

La **fonction d'erreur** est un des concepts les plus importants en *HCLP*. Puisque toutes les contraintes ne peuvent être satisfaites, il faut pouvoir évaluer, d'une manière ou d'une autre, l'importance de l'erreur commise en ne respectant pas certaines des contraintes non requises.

La fonction d'erreur doit avoir la propriété suivante:

- $e(c\theta) = 0$ si θ respecte la contrainte c

Parmi les fonctions d'erreurs possibles, citons:

- la fonction triviale $e(c\theta) = 0$ ou 1 (*predicate comparator*)
- une fonction qui utilise la métrique du domaine si celui-ci est métrique (*metric comparator*)
  - Ex: $erreur(X = Y) = dist(X,Y)$

Si $C = [c_1,...,c_k]$ est un vecteur de contraintes, $E(C\theta) = [e(c_1\theta),...,e(c_k\theta)]$

Une **séquence d'erreurs** est un vecteur de la forme $[E(H1\theta),...E(Hn\theta)]$ (une valeur de la fonction d'erreur par niveau)

Un **poids** peut être attribué à chaque contrainte (pour c_i: $w_i \in R^+$).

Il est parfois intéressant de combiner les erreurs par niveau avant de comparer les valuations. On utilise alors une **fonction d'agrégation** g pour des vecteurs de nombres réels[71].

La comparaison se fait via les relations:

- $<_g$ que l'on souhaite antiréflexive, antisymétrique et transitive;
- $<>_g$ que l'on souhaite réflexive et symétrique.

Il faut noter que « $<>_g$ » ne signifie pas nécessairement « = » dans la mesure où la relation n'est pas nécessairement transitive. Elle traduit simplement qu'il est impossible d'ordonner les éléments en utilisant « $<_g$ », ce qui est une nuance importante.

G est la généralisation de g aux séquences d'erreurs et fournit une séquence d'erreurs combinées.

- $R = [E(H_1\theta),...,E(H_n\theta)]$
- $G(R) = [g(E(H_1\theta)),...,g(E(H_n\theta))]$

Il est possible de définir un ordre lexicographique $<_{\boldsymbol{g}}$ sur les séquences d'erreurs combinées de la manière suivante:

$u_1, u_2,...,u_n <_{\boldsymbol{g}} v_1, v_2,...,v_n \Leftrightarrow \exists k \in 1...n\ \forall i, \in 1...k\text{-}1: u_i <>_g v_i \wedge u_k <_g v_k$

On peut finalement définir la **solution S à une hiérarchie de contraintes** de la manière suivante:

S_0 est l'ensemble des solutions possibles en n'examinant que les contraintes requises. Une solution[72] est une valuation de S_0 pour laquelle il n'existe pas de **meilleure** valuation dans S_0. Le qualificatif *meilleure* étant déterminé grâce à l'ordre lexicographique $<_{\boldsymbol{g}}$.

Plus formellement:

- $S_0 = \{\ \theta \mid \forall c \in H_0: e(c\theta)=0\ \}$
- $S = \{\ \theta \in S_0 \mid \forall \sigma \in S_0: \neg(G([E(H_1\sigma),...E(H_n\sigma)] <_{\boldsymbol{g}} G([E(H_1\theta),...E(H_n\theta)]))\ \}$

---

[71] Par exemple, g peut sommer les valeurs du vecteur, en choisir la plus grande,...

[72] À noter que nous parlons d'**une** solution et non de **la** solution.

L'ordre lexicographique assure que la priorité est donnée aux contraintes dont le niveau d'exigence est élevé.

Illustrons par un exemple tiré de la constitution des horaires.

- La session commence un lundi.
- Elle s'étend sur 15 jours.
- *Ne pas avoir d'examen le samedi après-midi* est une contrainte forte.
- *Ne pas avoir d'examen le samedi matin* est une contrainte faible.

Nous avons donc trois niveaux de contraintes:

H_0 (requises) $X > 0 \quad X < 62 \; X \neq 13 \; X \neq 14 \; X \neq 27 \; X \neq 28$

H_1 (fortes) $X \neq 12 \; X \neq 26$

H_2 (faibles) $X \neq 11 \; X \neq 25$

Considérons les deux valuations suivantes de X: θ_1 pour $X = 10$ et θ_2 pour $X = 11$. Les erreurs étant comptabilisables à partir du niveau 1 et utilisant un comparateur du type *predicate comparator*, on a:

au niveau 1	$e((X \neq 12)\theta_1) = 0$	$e((X \neq 26)\theta_1) = 0$	$E([(X \neq 12)\theta_1, (X \neq 26)\theta_1]) = [0,0]$
	$e((X \neq 12)\theta_2) = 0$	$e((X \neq 26)\theta_2) = 0$	$E([(X \neq 12)\theta_2, (X \neq 26)\theta_2]) = [0,0]$
au niveau 2	$e((X \neq 11)\theta_1) = 0$	$e((X \neq 25)\theta_1) = 0$	$E([(X \neq 11)\theta_1, (X \neq 25)\theta_1]) = [0,0]$
	$e((X \neq 11)\theta_2) = 1$	$e((X \neq 25)\theta_2) = 0$	$E([(X \neq 11)\theta_1, (X \neq 25)\theta_1]) = [1,0]$

Finalement,

$G(E([(X \neq 12)\theta_1, (X \neq 26)\theta_1]), E([(X \neq 11)\theta_1, (X \neq 25)\theta_1])) = [[0,0],[0,0]]$

précède, dans l'ordre lexicographique

$G(E([(X \neq 12)\theta_2, (X \neq 26)\theta_2]), E([(X \neq 11)\theta_2, (X \neq 25)\theta_2])) = [[0,0],[1,0]]$

et la valuation θ_1 est donc meilleure que la valuation θ_2.

6.3.4 Comparateurs

Le comparateur constitue un paramètre supplémentaire dans le schème *HCLP*. Son choix conditionne évidemment la production des solutions. On en distingue de trois catégories:

Les comparateurs locaux

Avec les comparateurs locaux, les erreurs ne sont pas globalisées. Chaque contrainte est considérée individuellement. θ_1 doit faire aussi bien que θ_2 pour toutes les contraintes des *k-1* premiers niveaux et, au niveau k, aussi bien pour toutes les contraintes et mieux pour au moins une de celles-ci.

Les comparateurs globaux

Les comparateurs globaux globalisent les erreurs au moyen d'une fonction g qui peut être de natures très différentes. Nous en donnons quelques exemples ci-dessous.

Les comparateurs régionaux

Les comparateurs régionaux ne globalisent pas les erreurs. Toutefois, des valuations incomparables à un certain niveau peuvent à nouveau être comparées au niveau inférieur. La discrimination est plus fine qu'avec un comparateur local.

Voici quelques exemples de comparateurs qui peuvent être utilisés et qui en montrent la variété. Nous décrivons la fonction de combinaison $\boldsymbol{g}$ utilisée et le sens donné aux relations $<>_g$ et $<_g$.

Parmi les comparateurs globaux, on trouve les comparateurs de type *weighted-sum-better*, *worst-case-better* et *least-squares-better*. On devine comment se fait la globalisation dans chacun des cas: somme

des erreurs, erreur maximum ou somme des carrés des erreurs. Dans chacun de ces cas, la relation $<>_g$ est équivalente à la relation d'égalité dans les réels et chacune des contraintes peut être affectée d'un poids.

Les comparateurs de type *locally-better* sont, comme le nom de leur catégorie l'indique, des comparateurs locaux. Il n'y a pas de globalisation et les opérateurs $<>_g$ et $<_g$ sont définis comme suit:

- $v_1 <_g v_2 \Leftrightarrow \forall i\, v_{1i} \leq v_{2i} \wedge \exists k\, v_{1k} < v_{2k}$
- $v_1 <> v_2 \Leftrightarrow \forall i\, v_{1i} = v_{2i}$

Les comparateurs de type *regionally-better* sont, comme le nom de leur catégorie l'indique, des comparateurs régionaux. La notion de solutions incomparables apparaît ici.

- $v_1 <_g v_2 \Leftrightarrow \forall i\, v_{1i} \leq v_{2i} \wedge \exists k\, v_{1k} < v_{2k}$
- $v_1 <> v_2 \Leftrightarrow \neg (v_1 <_g v_2 \vee v_2 <_g v_1)$

La combinaison de ces possibilités avec le choix d'une comparaison de type métrique ou prédicative fournit une grande variété de types de comparateurs parmi lesquels:

- weighted-sum-metric-better
- locally-metric-better
- worst-case-metric-better
- regionally-metric-better
- least-squares-metric-better
- locally-predicate-better
- ...

À noter que certaines combinaisons présentent peu d'intérêt comme:

- worst-case-predicate-better
- least-squares-predicate-better
- ...

Le fait que l'erreur soit toujours évaluée à 1 donne peu de sens à l'erreur maximum. Celle-ci correspond à 1 si au moins une des contraintes n'a pu être satisfaite et à 0 sinon. De même, réaliser la somme des carrés des erreurs dans le même contexte, revient à compter combien de contraintes n'ont pu être satisfaites.

6.3.5 Choix d'un comparateur

Le choix d'un comparateur reste une question difficile. Il existe actuellement peu de fondements théoriques à ce sujet. Plusieurs voies sont empruntées:

- existence d'un algorithme efficace;
- choix guidé par l'expérience (applications « interfaces utilisateurs »: *LPB*, applications de mise en forme graphique: *LSMB*,...);
- ...

Dans le cas du problème de la constitution des horaires de session, on imagine bien un comparateur de type *weighted-sum-predicate-better*. L'utilisation d'une métrique se justifie peu et plusieurs contraintes demandent qu'on attribue un poids plus important à leur « non-satisfaction ».

Dans la réalité de la constitution manuelle des horaires, on constate que certaines contraintes ne sont respectées que partiellement[73]. Le fait d'accepter des dérogations à certaines règles suffit généralement à débloquer le problème lorsque c'est le cas.

Utiliser une programmation basée sur une hiérarchie de contraintes et qui cherche à minimiser l'effet du non-respect de certaines de ces contraintes apparaît donc comme une piste intéressante à explorer.

Nous émettons toutefois plusieurs réserves. Le schème *HCLP* n'a pas été inspiré par les problèmes de constitution d'horaires, mais plutôt par des problèmes de gestion d'environnements graphiques. Sa grande « généricité » lui permet d'être envisagé dans des contextes aussi variés que la gestion d'environnements graphiques, l'édition et la mise en page de documents ou la planification. Rien ne dit cependant qu'il soit parfaitement adapté au type de problème qui nous intéresse. De plus, dans la problématique des horaires de sessions, et plutôt que des contraintes à hiérarchise, on trouve davantage de contraintes qui ne doivent pas toujours être respectées, en d'autres mots, des contraintes qui ne concernent qu'une partie des ressources et pas toutes.

À titre d'exemple, il est possible que certains professeurs ne souhaitent pas la précédence de l'écrit par rapport à l'oral. Cette observation peut difficilement être réglée en attribuant un quelconque niveau d'exigence à cette contrainte.

Il faut également signaler que les interpéteurs n'existent pas pour tous les comparateurs dont nous avons fait mention. Selon les informations dont nous disposons, il existe un interpréteur simple en *CLP(R)* pour *HCLP(R,LPB*[74]*)*. Il existe également un interpréteur écrit en *COMMON LISP* qui supporte plusieurs comparateurs métriques mais ce type de comparateurs nous est peu utile[75].

Une expérimentation dans cette direction demande donc une analyse préalable de l'ensemble des contraintes actuellement isolées (celles dont on tient compte et celles dont on ne tient compte que si c'est possible) afin d'envisager l'efficacité d'un développement au moyen d'un langage et d'interpréteurs permettant l'implantation de *HCLP*.

---

[73] Qu'il s'agisse d'horaires d'examens ou d'autres types d'horaires, la production de ceux-ci entraîne toujours la génération d'une frange d'utilisateurs mécontents, preuve par là, que certaines contraintes, explicitées ou non, ne sont pas respectées.

[74] LPB: Locally Predicate Better

[75] On trouve une description détaillée de ces interpréteurs dans [1].

Conclusion

Nos considérations seront de divers ordres.

La première observation, c'est que comme tout travail conséquent dans un domaine où l'expérience manque au départ, on est souvent amené à considérer que ce qui est acquis ne pourra être exploité que dans des projets futurs. En d'autres termes, un certain perfectionnisme pourrait nous pousser à vouloir tout recommencer, sachant ce que nous savons.

Nous avons le sentiment d'avoir donné à ce travail plusieurs dimensions assez différentes: le plus souvent, celle d'un travail appliqué, parfois celle d'un travail de recherche, parfois aussi celle d'un travail de compilation. Tout en rédigeant chaque chapitre, espérons-le, de manière lisible pour un public assez large, nous les avons destinés à des publics particuliers: celui des personnes qui se sont penchées sur ce type de problème, celui des utilisateurs (les plus concernés), celui des spécialistes de la programmation logique. Nous espérons que cette pluri-dimensionnalité n'en altère pas la qualité.

Concernant les espoirs à fonder sur l'utilité de ce travail et sur son avenir, nous avons l'ambition de l'améliorer dans les limites de ce que nous proposons dans les paragraphes 1 et 2 du chapitre 6 consacré aux perspectives. Nous souhaitons aussi le tester et le rendre accessible à d'autres facultés plus peuplées.

Notre objectif était avant tout de juger de la faisabilité du projet. Outre le fait que celui-ci a fourni des solutions acceptables sur des données réelles, il nous a permis de constater que le chemin conduisant à des améliorations était large et tout à fait carrossable, essentiellement pour deux raisons:

- la modélisation des données est très ouverte,
- un développement incrémental est possible.

Le bon contact que nous avons avec les utilisateurs experts nous permet d'envisager d'autres discussions susceptibles d'améliorer encore l'outil existant. Nous pensons essentiellement travailler sur la concentration des oraux des professeurs et sur l'introduction de nouveaux paramètres pour les cours. Il restera toujours une difficulté, celle de pouvoir déterminer (pour ne pas dire juger) ce qui est réellement de l'ordre de la contrainte.

L'intérêt du programme est lié à la simplicité et la transparence de la gestion des données, à la vitesse à laquelle les résultats sont produits (quelques secondes) et à la production d'horaires individuels pour les étudiants.

Nous formulons cependant la recommandation que l'équipe de conception des horaires puisse disposer, localement et assez rapidement dans l'année, des informations concernant l'inscription des étudiants au cours, de manière à pouvoir les mettre à jour en permanence et en fonction d'autres informations dont cette équipe dispose (reports de notes, inscriptions panachées, abandons,...).

Quant à la piste des hiérarchies de contraintes, elle nous paraît prématurée en l'absence d'expérience d'application de ce schème à ce type de problème et compte tenu des bons résultats produits par le schème *CLP*.

Bibliographie

[1] **Hierarchical Constraint Logic Programming**
Molly Ann Wilson
Ph.D. dissertation, Department of Computer Science and Engineering, University of Washington
April 1993.

[2] **Timetabling in Constraint Logic Programming**
Maria Kambi, David Gilbert
Department of Computer Science, The City University London
October 1996.

[3] **University Timetabling using Constraint Handling Rules**
Slim Abdennadher, Michael Marte
Institut für Informatik München
April 1998

[4] **On Methods of Constraint-Based Timetabling**
Hans-Joachim Goltz
German National Research Center for Information Technology Berlin
April 2000

[5] **A Memetic Algorithm for University Exam Timetabling**
Edmund K. Burke, James P. Newall, Rupert F. Weare
Department of Computer Science, University of Nottingham
September 1995

[6] **Abstracting Soft Constraint**
S. Bistarelli, P. Codognet, Y. Georget, F. Rossi
1999

[7] **Constraint-based Timetabling with Student Schedules**
Hana Rudová and Ludek Matyska
Faculty of Informatics, Masaryk University Brno
August 2000

[8] **Local Search Techniques for Educational Timetabling Problems**
Andrea Shaerf, Luca Di Gaspero
Universita di Udine
2001

[9] **The GNU Prolog System and its implementation**
Daniel Diaz, Philippe Codognet

[10] **Tabu Search Techniques for Examination Timetabling** (abstract)
L. Di Gaspero, A. Schaerf
Universita di Udine
2001

[11] **Constraint-based Timetabling – a Case study**
A. M. Abbas, E. P. K. Tsang

[12] **Tabu search techniques for large high-school timetabling problems**
A Schaerf
Computer Science/Department of Interactive Systems
Amsterdam 1996

[13] **Examination Timetabling**
The Temposcope: a Computer Instrument for the Idealist Timetabler (full paper)
M. Beynon, A. Ward, S. Maad, A. Wong, S. Rasmequan, S. Russ (Coventry, UK)

[14] **Multicriteria Approach to Timetabling Problems (abstract)**
E. K. Burke, Y. Bykov, S. Petrovic (Nottingham, UK)

[15] **Examination Timetabling Using Set Variables (abstract)**
L. P. Reis, E. Oliveira (Porto, Portugal)

[16] **Examination Timetables and Tabu Search With Longer Term Memory (full paper)**
G. M. White, B. S. Xie (Ottawa, Canada)

[17] **Guide to Constraint Programming**
R. Bartak 1998
http://kti.mff.cuni.cz/~bartak/constraints/

ANNEXES

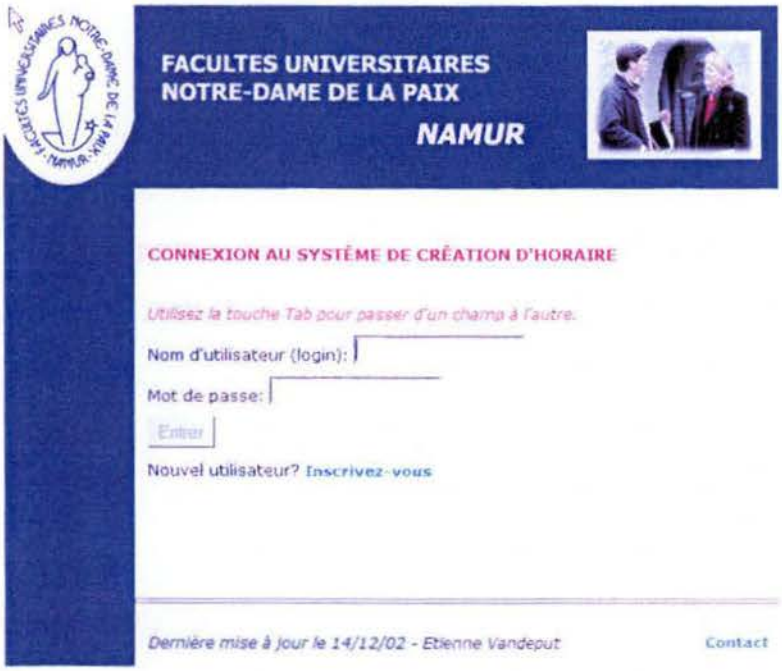
Annexe 1

Interface graphique de saisie des données

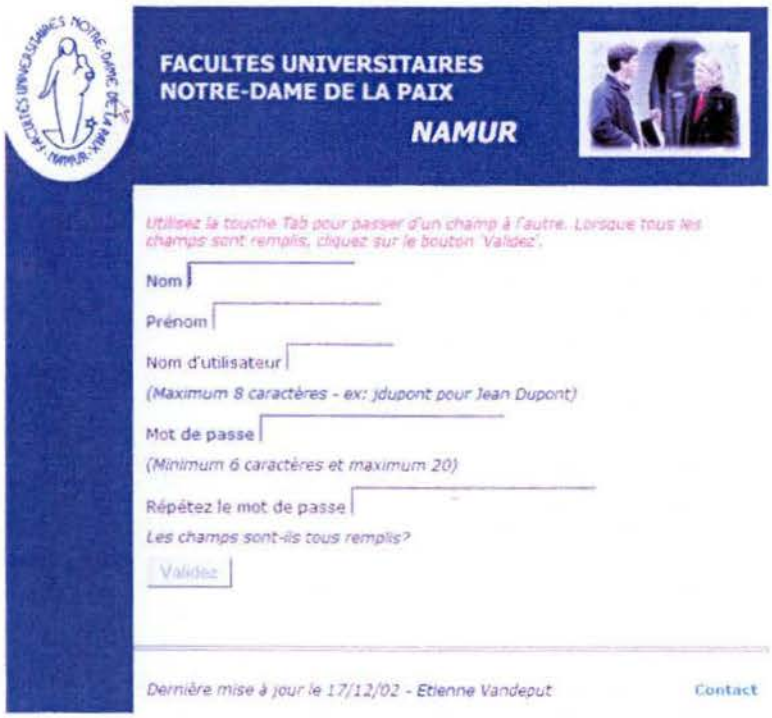
Nous proposons un ensemble significatif des écrans de communication entre l'utilisateur expert et la base de données locale. Ces écrans sont des pages Web dynamiques générées en *PHP*. La base de données est une base de données *MySQL* dont la structure est décrite dans le document.

Ecran de login

L'utilisateur se connecte classiquement en fournissant son identificateur et un mot de passe. S'il s'agit d'un nouvel utilisateur, un lien lui permet d'accéder à une page d'inscription.



Ecran de création d'un nouvel utilisateur expert



La correction des données saisies à travers ces formulaires est contrôlée (fourniture d'identificateur inexistant, mot de passe trop court, identifiant trop long ou existant déjà,…) et génère divers écrans signalant le problème. En voici trois exemples illustratifs parmi d'autres.

Mauvais identifiant:

SESSION DE TRAVAIL

Utilisateur: **vandeput**

ERREUR: Cet utilisateur n'existe pas.

Nouvel essai

Mauvais mot de passe:

SESSION DE TRAVAIL

Utilisateur: **eva**

Nom complet: **Etienne Vandeput**

ERREUR: Le mot de passe n'est pas valide.

Nouvel essai

Mot de passe mal reproduit:

Utilisez la touche Tab pour passer d'un champ à l'autre. Lorsque tous les champs sont remplis, cliquez sur le bouton 'Validez'.

Nom Vandeput

Prénom Etienne

Nom d'utilisateur eva

(Maximum 8 caractères - ex: jdupont p

Mot de passe ******

(Minimum 6 caractères et maximum 20)

Répétez le mot de passe ******

Les champs sont-ils tous remplis?

Validez

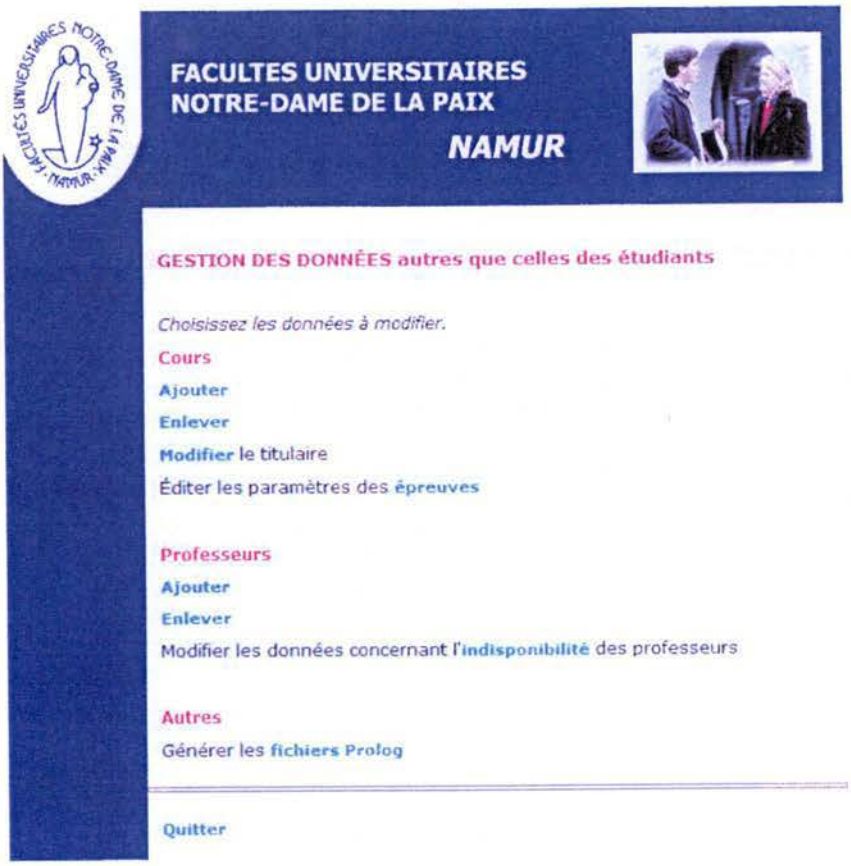
[Application JavaScript]

Les mots de passe fournis ne sont pas identiques!

OK

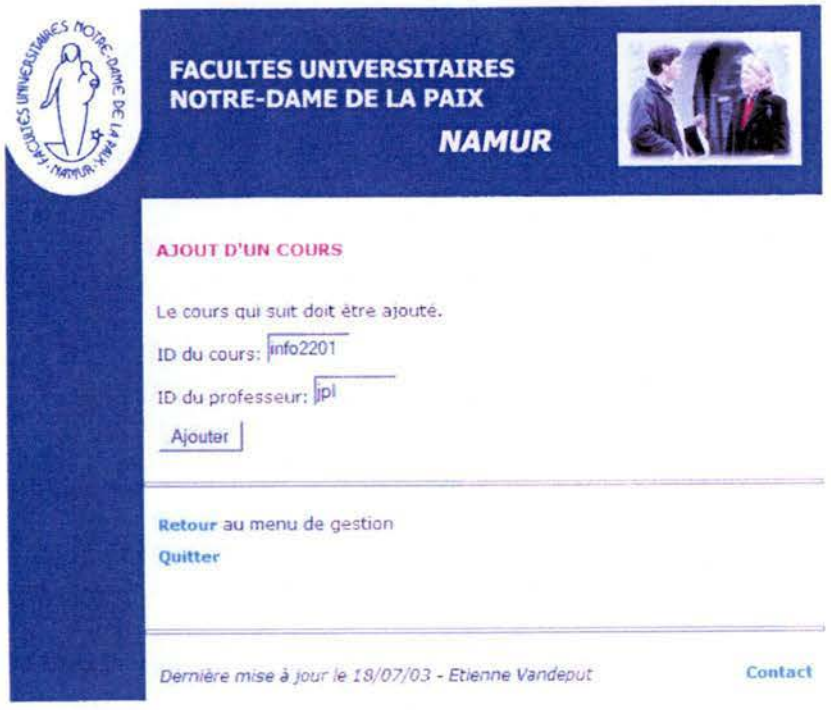
Dernière mise à jour le 17/12/02 - Etienne Vandeput **Contact**

Écran de gestion des données

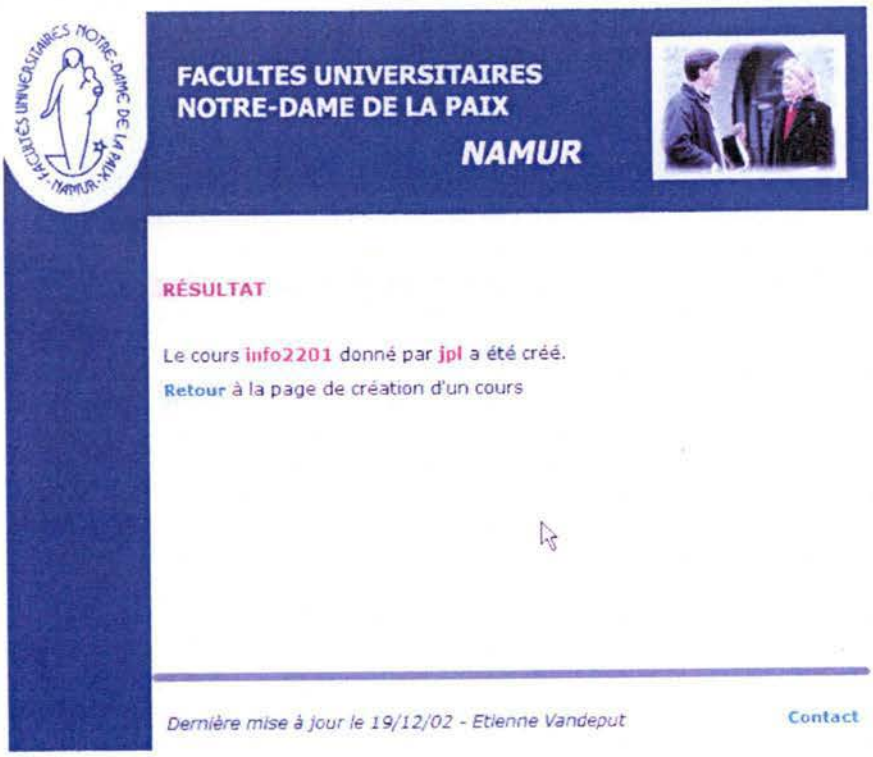


Écrans d'ajout d'un cours

L'ajout des cours se fera sans doute en une seule fois et cette option ne sera plus utilisée qu'à de très rares occasions.



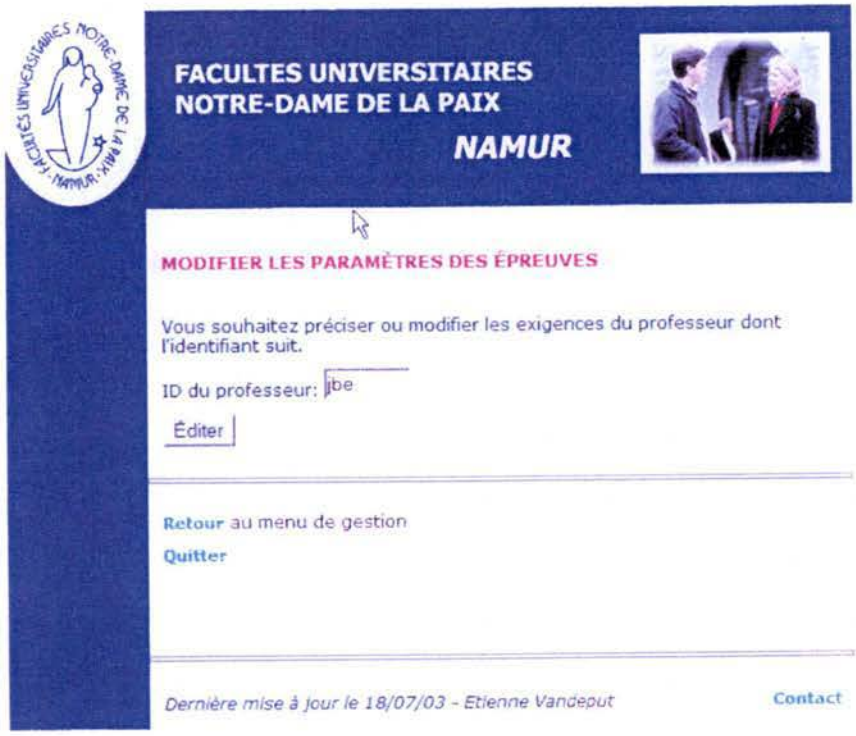
La fourniture d'un ID professeur conduit à un écran fournissant un feed-back.



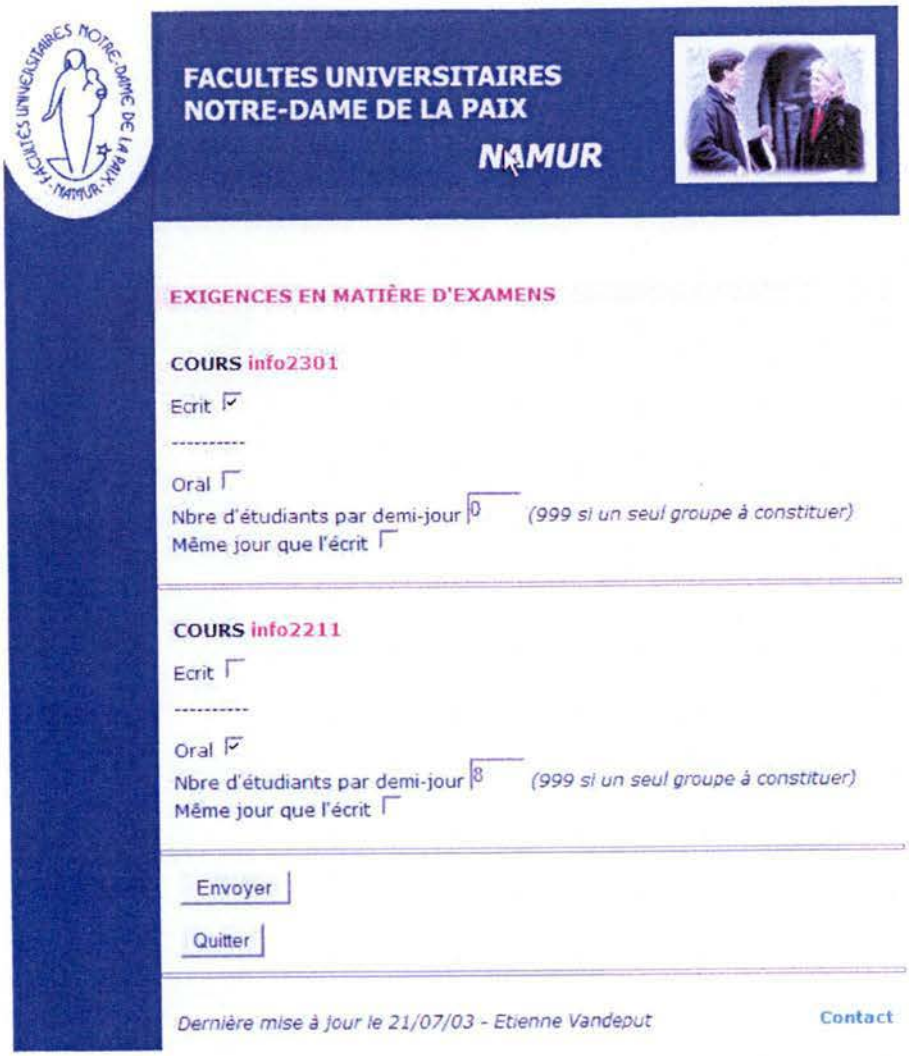
Les écrans de suppression d'un professeur de la base de données et de changement d'attribution sont très semblables et seront sans doute peu fréquemment utilisés.

Écrans de modification des paramètres d'un cours

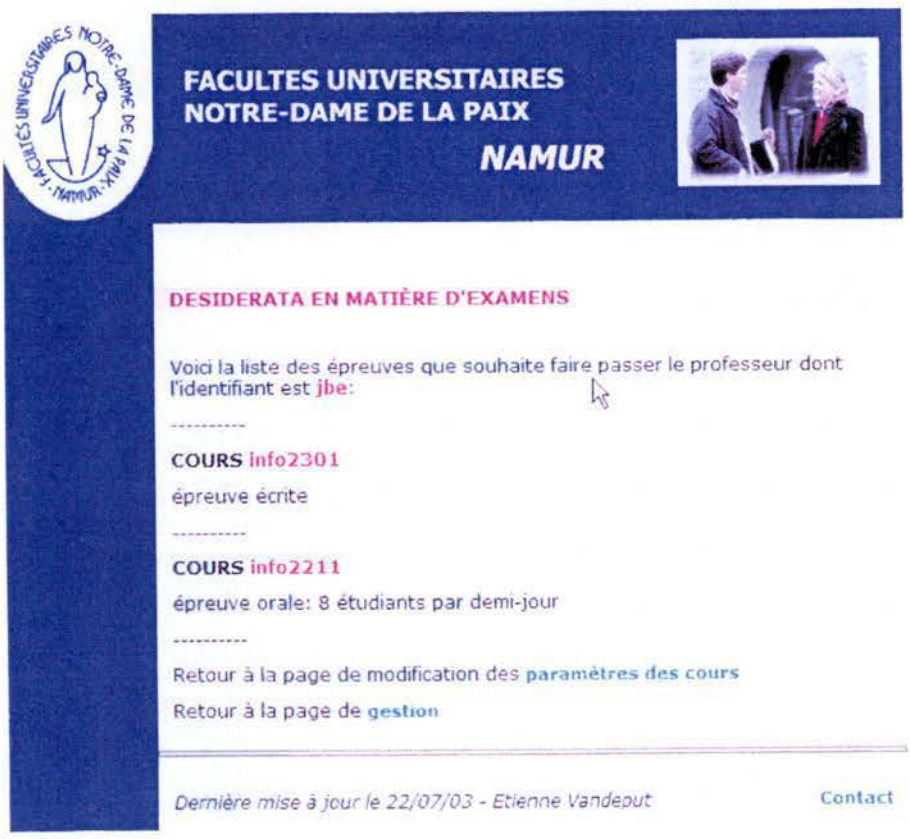
Des quatre items de la rubrique *Cours*, c'est l'option qui sera la plus utilisée.



Tous les cours du professeur sont affichés et leurs paramètres peuvent être modifiés.

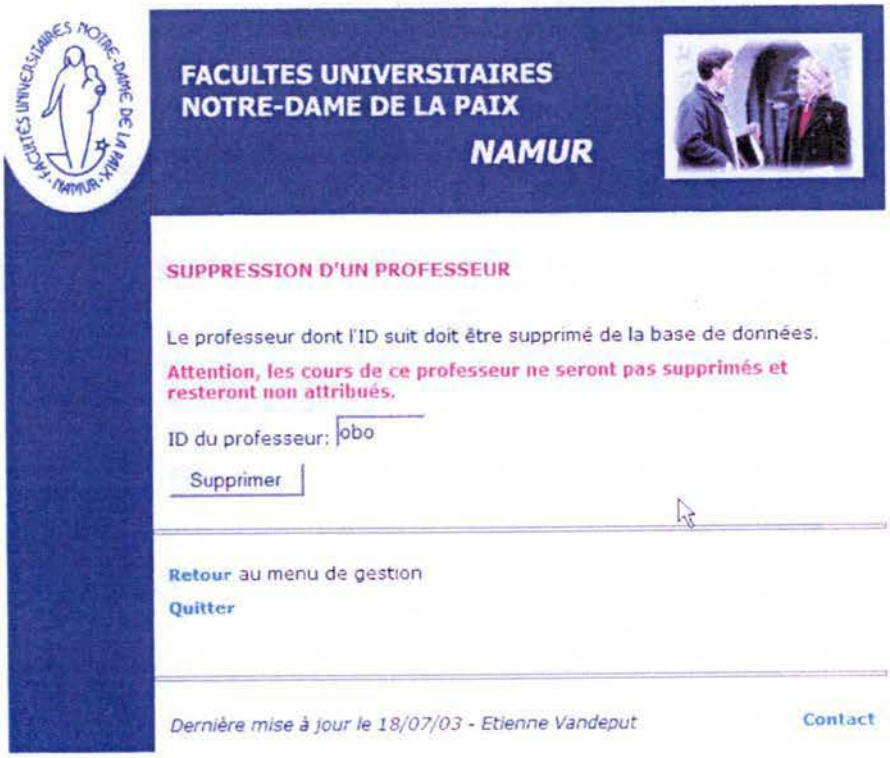


L'envoi des données donne lieu à un récapitulatif pour une vérification éventuelle (ou une impression).

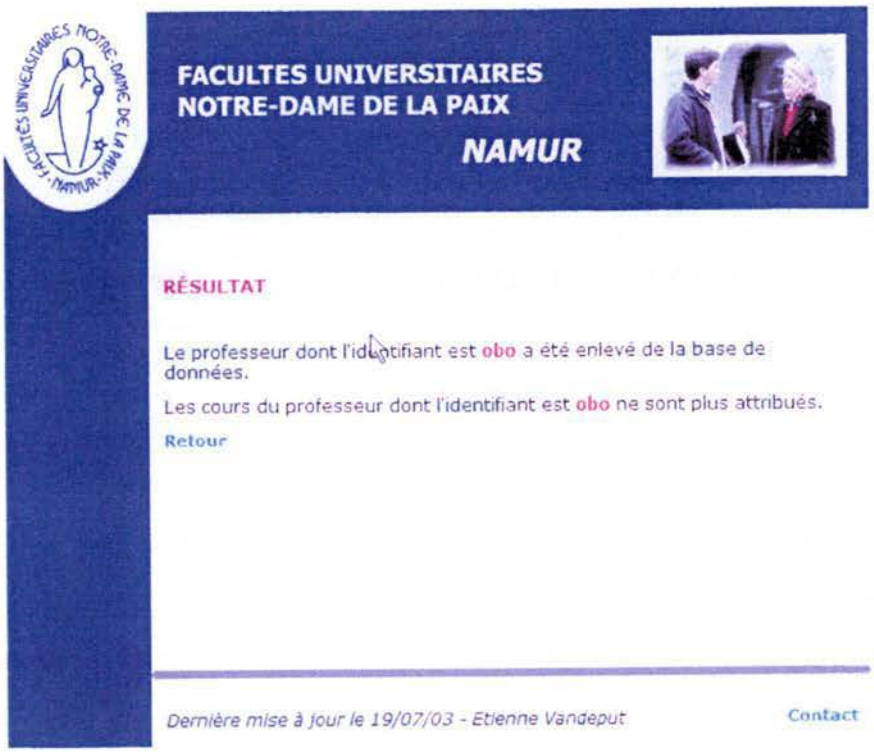


Écrans des données concernant les professeurs

Les écrans d'ajout et de suppression sont assez semblables et classiques. Nous présentons ci-dessous l'écran de suppression.



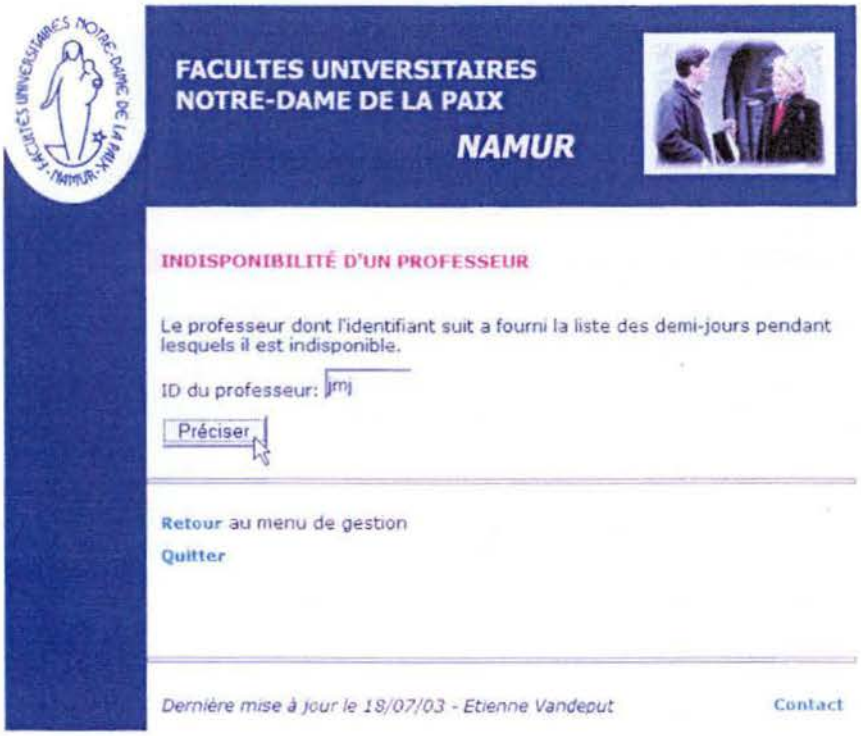
...et le feed-back associé.



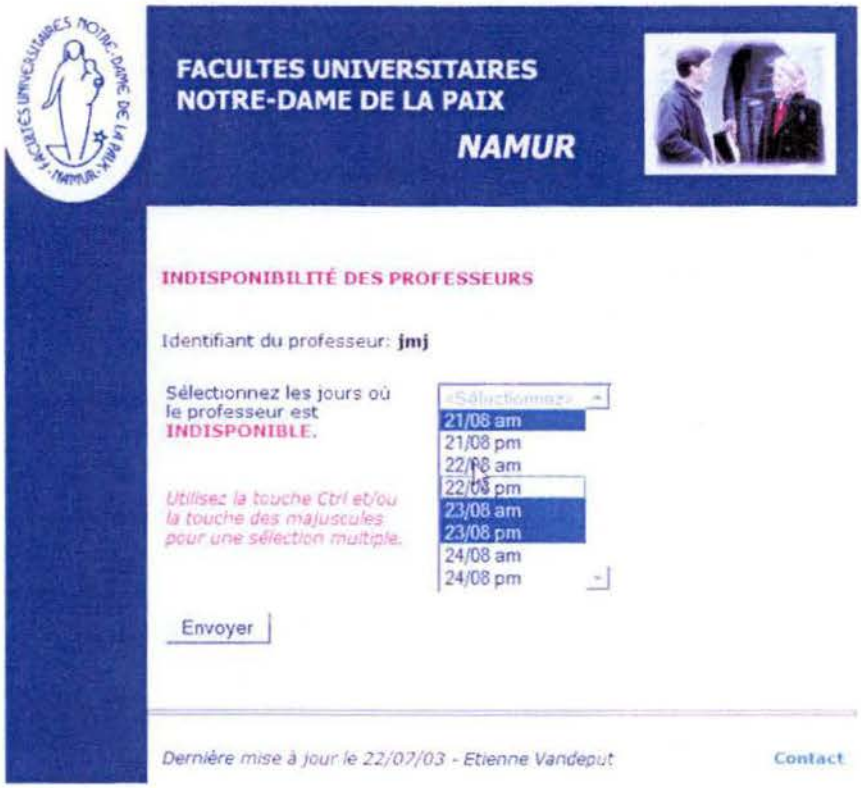
Le cas pathologique de l'identifiant incorrect est évidemment prévu.

Les écrans pour la gestion des indisponibilités sont:

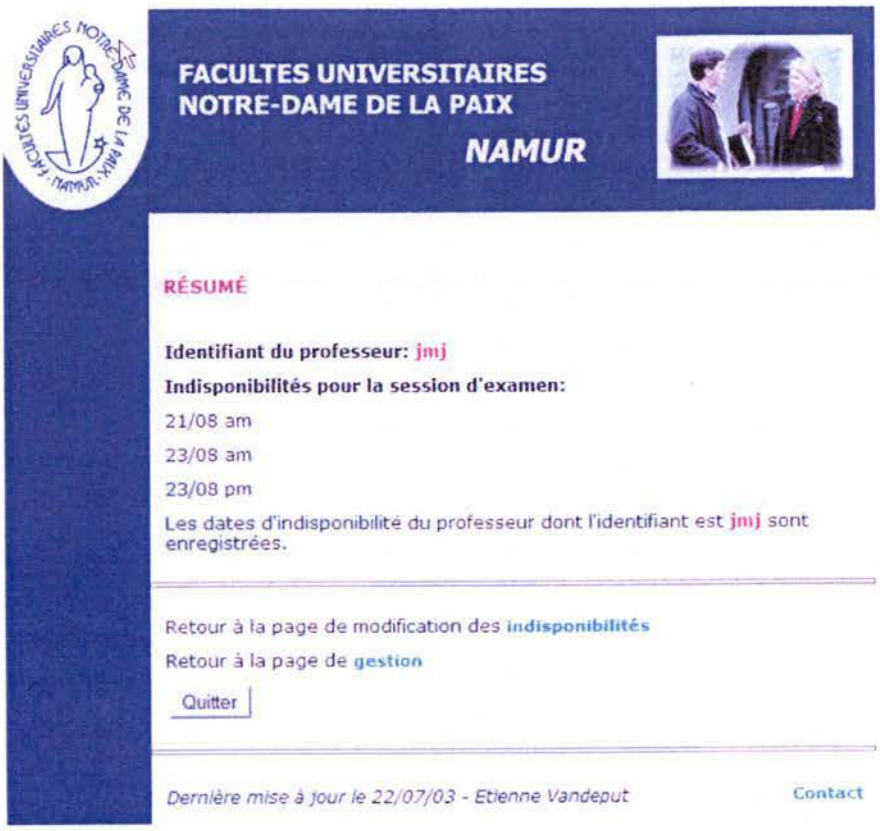
- un écran de saisie de l'identifiant du professeur,



- un écran pour préciser ses indisponibilités dans une liste déroulante,



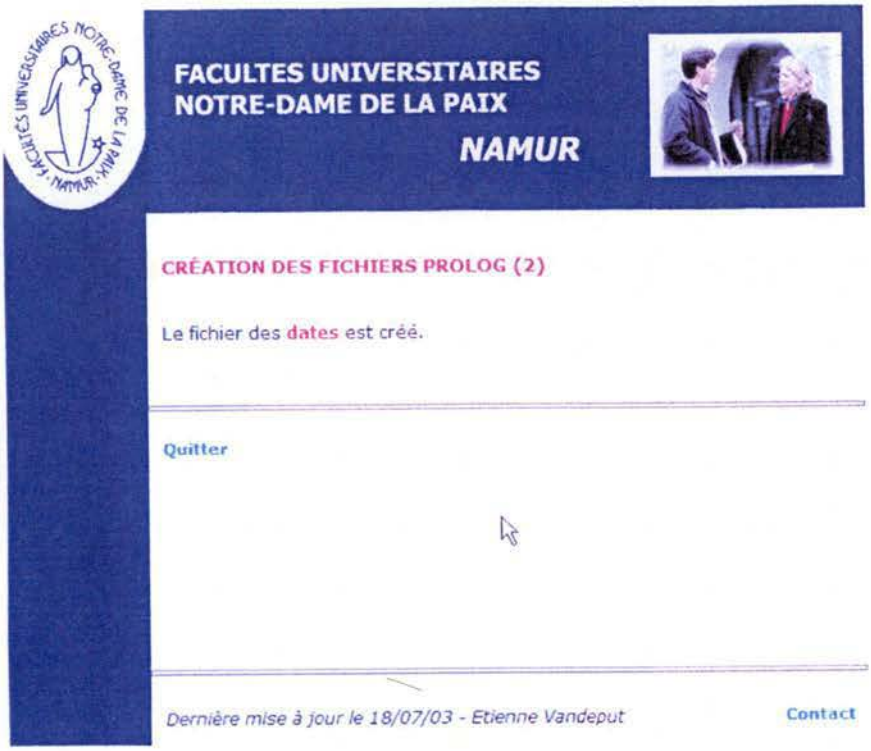
- un écran de feed-back.



Écrans pour la génération des fichiers Prolog

Cette génération est relativement transparente. L'utilisateur est averti de la création des fichiers prêts à être utilisés par le programme. Lors du premier feed-back, l'utilisateur est invité à préciser d'autres paramètres, et notamment ceux qui permettront la génération du fichier des dates.

L'envoi de ces dernières données est suivi du feed-back suivant:



Reste à l'utilisateur à faire exécuter le programme.

Annexe 2

Aperçu des fichiers Prolog traités par le programme

Les données fournies à travers l'interface web sont encapsulées dans des fichiers sous une forme très spécifique. Voici un exemple de chacun de ces fichiers correspondant à la situation de juin 2003.

Fichier des attributions (profs.txt)

```
[  enseigne( ade, [ info2114 , info2328 ]),
   enseigne( ble, [ info2109 ]),
   enseigne( clo, [ info2102, info2304 ]),
   enseigne( cvw, [ sent2288 ]),
   enseigne( emo, [ info2322 ]),
   enseigne( fbo, [ info2233 ]),
   enseigne( gde, [ info2327 ]),
   enseigne( gva, [ info2112 ]),
   enseigne( jbe, [ info2211, info2301 ]),
   enseigne( jfi, [ info2101 ]),
   enseigne( jgh, [ info2223 ]),
   enseigne( jlh, [ info2106,  info2226 ]),
   enseigne( jmj, [ info2209 ]),
   enseigne( jnc, [ info2110 ]),
   enseigne( jpc, [ info2302 ]),
   enseigne( jpl, [ info2201, info2220, info2222 ]),
   enseigne( jra, [ info2229, info2230 ]),
   enseigne( jva, [ ielv2212, ielv2113, ielv2307 ]),
   enseigne( lsc, [ info2231 ]),
   enseigne( mno, [ info2105, info2202, info2221, info2320, info2321 ]),
   enseigne( nha, [info2107, info2204, info2205, info2324 ]),
   enseigne( obo, [ info2210 ]),
   enseigne( pre, [ info2111 ]),
   enseigne( pys, [ info2108, info2326, info3101 ]),
   enseigne( ven, [ info2208, info2232, info2305, info2325 ])].
```

Fichier des épreuves (epr.txt)

```
[  epreuve(info2211,ecrit+oral(12)),
   epreuve(info2301,ecrit+oral(9)),
   epreuve(info2233,oral(9)),
   epreuve(info2210,oral(999)),
   epreuve(info2302,oral(999)),
   epreuve(info2110,oral(12)),
   epreuve(info2114,oral(999)),
   epreuve(info2327,oral(9)),
   epreuve(info2203,oral(999)),
   epreuve(info2208,oral(12)),
   epreuve(info2232,ecrit),
   epreuve(info2325,oral(20)),
   epreuve(info2323,oral(999)),
```

```
  epreuve(info2223,oral(999)),
  epreuve(info2107,ecrit),
  epreuve(info2204,ecrit),
  epreuve(info2205,oral(999)),
  epreuve(info2324,oral(9)),
  epreuve(info2106,ecrit),
  epreuve(info2226,ecrit),
  epreuve(info2209,ecrit),
  epreuve(info2109,oral(9)),
  epreuve(info2101,ecrit),
  epreuve(info2201,ecrit),
  epreuve(info2222,ecrit),
  epreuve(info2220,oral(999)),
  epreuve(info2303,ecrit),
  epreuve(info2224,ecrit),
  epreuve(info2102,oral(9)),
  epreuve(info2304,oral(999)),
  epreuve(info2322,oral(12)),
  epreuve(info2105,ecrit+oral(8)),
  epreuve(info2221,oral(999)),
  epreuve(info2202,ecrit*oral),
  epreuve(info2320,oral(999)),
  epreuve(info2321,oral(999)),
  epreuve(info2230,oral(6)),
  epreuve(info2229,ecrit),
  epreuve(info2306,ecrit),
  epreuve(info2111,ecrit+oral(9)),
  epreuve(info2108,ecrit),
  epreuve(info3101,oral(9)),
  epreuve(info2326,oral(9)),
  epreuve(info2231,oral(999)),
  epreuve(info2112,oral(9)),
  epreuve(ielv2113,oral(999)),
  epreuve(ielv2212,oral(999)),
  epreuve(ielv2307,oral(999)),
  epreuve(ielv2113,oral(999)),
  epreuve(ielv2212,oral(999)),
  epreuve(ielv2307,oral(999)),
  epreuve(sent2288,ecrit) ].
```

Fichier des indisponibilités (indis.txt)

```
[
indispo(jbe,
[1,2,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38]),
indispo(fbo,[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24, 25,26,27,28,29,30,31,32]),
indispo(obo, []),
```

```
indispo(jnc, [10,11,12,13,14,15,16,17,18,19,20,21,22]),
indispo(ven, [1,2,3,4,5,6,19,20,21,25,26]),
indispo(jfi, []),
indispo(nha, [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]),
indispo(jlh, []),
indispo(jmj, [9,10]),
indispo(ble, []),
indispo(jpl, [1,2,3,4,5,6,25,26,29,30]),
indispo(clo, [1,2,3,4,5,6,15,16,25,26,29,30,31,32,33,34]),
indispo(mno, [1,2,3,4,29,30,31,32,33,34,35,36,37,38]),
indispo(jra, [9,10,11,12,13,14,15,16,21,22]),
indispo(pre, [1,2,3,4,5,6,9,10,15,16]),
indispo(pys, [7,8,9,10,11,12,13,14,15,16,17,18,19,20]),
indispo(gva, [15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32]),
indispo(jva, [1,2]),
indispo(cvw, [])].
```

Fichier des inscriptions (inscr.txt)

```
[inscrit(achbany,[info2229,ielv2212,info2201,info2202,info2205,info2208,i
nfo2210,info2211,info2226,info2230,info2232,info2233]),
inscrit(anciaux,[ielv2212,info2201,info2202,info2205,info2208,info2210,in
fo2211,info2221,info2224,info2230,info2232,info2233]),
inscrit(andre,[ielv2113,info2101,info2102,info2105,info2106,info2107,info
2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(arman,[info2229,ielv2212,info2201,info2202,info2205,info2208,info
2210,info2211,info2224,info2230,info2232,info2233]),
inscrit(barras,[ielv2212,info2201,info2202,info2205,info2208,info2210,inf
o2211,info2223,info2224,info2230,info2232,info2233]),
inscrit(barthelemy,[ielv2212,info2201,info2202,info2205,info2208,info2210
,info2211,info2221,info2226,info2230,info2232,info2233]),
inscrit(baudoin,[info2101,info2105,info2106,info2107,info2108,info2109,in
fo2110,info2114,sent2288]),
inscrit(benko,[ielv2212,info2201,info2202,info2205,info2208,info2210,info
2211,info2224,info2230,info2231,info2232,info2233]),
inscrit(biekonda_lonto,[ielv2113,info2101,info2102,info2105,info2106,info
2107,info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(bol,[info2229,ielv2212,info2201,info2202,info2205,info2208,info22
10,info2211,info2224,info2230,info2232,info2233]),
inscrit(boursin,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2324,info2327]),
inscrit(briquet,[info2201,info2204,info2327,info2301,info2322,info2302,in
fo2306,ielv2307]),
inscrit(buyle,[info2322,ielv2307,info2301,info2302,info2303,info2306,info
2325,info2327,info2328]),
inscrit(canlas,[ielv2212,info2201,info2202,info2205,info2208,info2210,inf
o2211,info2221,info2224,info2230,info2232,info2233]),
inscrit(capelle,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2324,info2326,info2328]),
inscrit(carouy,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2112,sent2288]),
```

```
inscrit(cleve,[info2229,ielv2212,info2201,info2202,info2205,info2208,info
2210,info2211,info2224,info2226,info2232,info2233]),
inscrit(colot,[ielv2212,info2201,info2202,info2205,info2208,info2210,info
2211,info2224,info2230,info2231,info2232,info2233]),
inscrit(coquiart,[ielv2212,info2201,info2202,info2205,info2208,info2210,i
nfo2211,info2222,info2224,info2230,info2232,info2233]),
inscrit(cornet_d_elzius,[info2322,ielv2307,info2301,info2302,info2303,inf
o2306,info2326,info2327,info2328]),
inscrit(coste,[info2322,ielv2307,info2301,info2302,info2303,info2306,info
2324,info2325,info2328]),
inscrit(custinne,[ielv2212,info2201,info2202,info2205,info2208,info2210,i
nfo2211,info2221,info2223,info2230,info2232,info2233]),
inscrit(cuvellier,[info2322,ielv2307,info2301,info2302,info2303,info2306,
info2326,info2328,info3101]),
inscrit(d_hollander,[info2322,ielv2307,info2301,info2302,info2303,info230
6,info2324,info2327,info2328]),
inscrit(dallons_g,[info2229,ielv2212,info2201,info2202,info2205,info2208,
info2210,info2211,info2226,info2230,info2232,info2233]),
inscrit(dallons_q,[info2322,ielv2307,info2301,info2302,info2303,info2306,
info2324,info2325,info2328]),
inscrit(de_coster,[info2322,ielv2307,info2301,info2302,info2303,info2306,
info2326,info2327,info2328]),
inscrit(de_gols,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2326,info2327,info2328]),
inscrit(de_menten_de_horne,[info2229,ielv2212,info2201,info2202,info2205,
info2208,info2210,info2211,info2222,info2226,info2232,info2233]),
inscrit(de_mey,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2221,info2224,info2232,info2233]),
inscrit(de_vreese,[info2229,ielv2212,info2201,info2202,info2205,info2208,
info2210,info2211,info2222,info2223,info2232,info2233]),
inscrit(decock,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2224,info2230,info2232,info2233]),
inscrit(decostre,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2324,info2327,info2328]),
inscrit(defat,[info2229,ielv2212,info2201,info2202,info2205,info2208,info
2210,info2211,info2223,info2224,info2232,info2233]),
inscrit(delange,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(deliege,[info2229,ielv2212,info2201,info2202,info2205,info2208,in
fo2210,info2211,info2226,info2230,info2232,info2233]),
inscrit(denayer,[info2229,ielv2212,info2201,info2202,info2205,info2208,in
fo2210,info2211,info2224,info2226,info2232,info2233]),
inscrit(diet,[info2322,ielv2307,info2301,info2302,info2303,info2306,info2
324,info2327,info2328]),
inscrit(donckels,[info2101,info2106,info2109,info2107]),
inscrit(donnet,[info2322,ielv2307,info2301,info2302,info2303,info2306,inf
o2324,info2327,info2328]),
inscrit(dosquet,[ielv2212,info2201,info2202,info2205,info2208,info2210,in
fo2211,info2221,info2224,info2230,info2232,info2233]),
inscrit(dubois_x,[ielv2113,info2101,info2102,info2105,info2106,info2107,i
nfo2108,info2109,info2110,info2111,info2112,sent2288]),
```

```
inscrit(ducochez,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2324,info2326,info2328]),
inscrit(duterme,[ielv2212,info2201,info2202,info2205,info2208,info2210,in
fo2211,info2224,info2226,info2230,info2232,info2233]),
inscrit(fabry,[ielv2212,info2201,info2202,info2205,info2208,info2210,info
2211,info2224,info2226,info2230,info2232,info2233]),
inscrit(foulard,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2326,info2327,info2328]),
inscrit(francois,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2325,info2327,info2328]),
inscrit(frippiat,[ielv2113,info2101,info2102,info2105,info2106,info2107,i
nfo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(gasore_ndindabahizi,[info2101,info2105,info2106,info2108,info2109
]),
inscrit(gengler,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(genicot,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(gilson,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(gobert,[ielv2307,info2301,info2302,info2303,info2306,info2326,inf
o2327,info2328,info3101]),
inscrit(gregoire,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2324,info2325,info2328]),
inscrit(gries,[ielv2212,info2201,info2202,info2205,info2208,info2210,info
2211,info2224,info2226,info2230,info2232,info2233]),
inscrit(grimard,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2324,info2327,info2328]),
inscrit(gruselin,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2324,info2326,info2328]),
inscrit(hallez,[info2322,ielv2307,info2301,info2302,info2303,info2306,inf
o2323,info2324,info2328]),
inscrit(hamoir,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(hick,[info2201,info2204,info2304,info2302,info2306]),
inscrit(hynderick_de_ghelcke,[ielv2212,info2201,info2202,info2205,info220
8,info2210,info2211,info2224,info2226,info2231,info2232,info2233]),
inscrit(jadoul,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2112,sent2288]),
inscrit(jadoulle,[ielv2212,info2201,info2202,info2205,info2208,info2210,i
nfo2211,info2224,info2226,info2230,info2232,info2233]),
inscrit(jamart,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(jasselette,[ielv2113,info2101,info2102,info2105,info2106,info2107
,info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(jeanpierre,[info2303,ielv2113,info2101,info2102,info2105,info2106
,info2107,info2108,info2109,info2110,info2114,sent2288]),
inscrit(jeunejean,[info2229,ielv2212,info2201,info2202,info2205,info2208,
info2210,info2211,info2224,info2226,info2232,info2233]),
inscrit(lambot,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2224,info2226,info2232,info2233]),
```

```
inscrit(le_fevere_de_ten_hove,[info2229,ielv2212,info2201,info2202,info22
05,info2208,info2210,info2211,info2230,info2231,info2232,info2233]),
inscrit(le_kim,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(leclercq_a,[info2322,ielv2307,info2301,info2302,info2303,info2306
,info2324,info2327,info2328]),
inscrit(lefevre,[info2303,ielv2113,info2101,info2102,info2105,info2106,in
fo2107,info2108,info2109,info2110,info2114,sent2288]),
inscrit(lejeune,[ielv2212,info2201,info2202,info2205,info2208,info2210,in
fo2211,info2223,info2224,info2226,info2232,info2233]),
inscrit(loffet,[ielv2307,info2322,info2301,info2302,info2303,info2306,inf
o2326,info2327,info2328]),
inscrit(mabika_kaningu,[ielv2212,info2201,info2202,info2205,info2208,info
2210,info2211,info2221,info2222,info2223,info2232,info2233]),
inscrit(maes,[info2101,info2201,info2209,info2232,info2204,info2221,info2
202,info2211,info2208,info2233]),
inscrit(magnant,[ielv2212,info2201,info2202,info2205,info2208,info2210,in
fo2211,info2221,info2224,info2230,info2232,info2233]),
inscrit(mainil,[info2303,ielv2113,info2101,info2102,info2105,info2106,inf
o2107,info2108,info2109,info2110,info2114,sent2288]),
inscrit(marquet,[info2229,ielv2212,info2201,info2202,info2205,info2208,in
fo2210,info2211,info2223,info2226,info2232,info2233]),
inscrit(mathieu,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2326,info2327,info2328]),
inscrit(mercier,[info2105,info2106]),
inscrit(miche_g,[info2229,ielv2212,info2201,info2202,info2205,info2208,in
fo2210,info2211,info2230,info2231,info2232,info2233]),
inscrit(miche_m,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(moyen,[info2107,info2108,info2105,sent2288,info2201,info2232]),
inscrit(mysliwiec,[ielv2307,info2322,info2301,info2302,info2303,info2306,
info2324,info2327]),
inscrit(nkezabera,[ielv2212,info2201,info2202,info2205,info2208,info2210,
info2211,info2224,info2226,info2230,info2232,info2233]),
inscrit(noel,[info2229,ielv2212,info2201,info2202,info2205,info2208,info2
210,info2211,info2224,info2226,info2232,info2233]),
inscrit(nolard,[info2322,ielv2307,info2301,info2302,info2303,info2306,inf
o2325,info2327,info2328]),
inscrit(nsimba_matondo,[info2106,info2107,info2105,info2108,info2208,info
2209]),
inscrit(patris,[info2322,ielv2307,info2301,info2302,info2303,info2306,inf
o2326,info2327,info2328]),
inscrit(picard,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2224,info2226,info2232,info2233]),
inscrit(plichart,[info2322,ielv2307,info2301,info2302,info2303,info2306,i
nfo2324,info2327,info2328]),
inscrit(ponsen,[info2201,info2232,info2209,info2202,info2226,info2211,inf
o2301,info2302,info2306]),
inscrit(pouplard,[info2229,ielv2212,info2201,info2202,info2205,info2208,i
nfo2210,info2211,info2224,info2230,info2232,info2233]),
inscrit(purnelle,[ielv2113,info2101,info2102,info2105,info2106,info2107,i
nfo2108,info2109,info2110,info2111,info2112,sent2288]),
```

```
inscrit(raes,[info2229,ielv2212,info2201,info2202,info2205,info2208,info2
210,info2211,info2230,info2231,info2232,info2233]),
inscrit(ramdoyal,[ielv2307,info2322,info2301,info2302,info2303,info2306,i
nfo2324,info2327,info2328]),
inscrit(randolet,[ielv2212,info2201,info2202,info2205,info2208,info2210,i
nfo2211,info2221,info2223,info2230,info2232,info2233]),
inscrit(renaud,[info2301,info2224,info2226,info2221,info2229,info2306,inf
o2204,info2205,info2208,info2210,info2232]),
inscrit(reygaerdts,[info2322,ielv2307,info2301,info2302,info2303,info2306
,info2326,info2327,info2328]),
inscrit(riquet,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(s_jongers,[info2201,info2204,info2222,info2322,info2302,info2306,
ielv2307]),
inscrit(sandron,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(schuurman,[ielv2113,info2101,info2102,info2105,info2106,info2107,
info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(sevrin,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(solheid,[ielv2113,info2101,info2102,info2105,info2106,info2107,in
fo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(stalens,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2324,info2326,info2328]),
inscrit(stouffs,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2326,info2327,info2328]),
inscrit(struyven,[ielv2113,info2101,info2102,info2105,info2106,info2107,i
nfo2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(teller,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2226,info2230,info2232,info2233]),
inscrit(thilly,[ielv2212,info2201,info2202,info2205,info2208,info2210,inf
o2211,info2222,info2224,info2230,info2232,info2303,info2328,info2233]),
inscrit(toussaint,[info2229,ielv2212,info2201,info2202,info2205,info2208,
info2210,info2211,info2226,info2230,info2232,info2233]),
inscrit(trifin,[info2229,ielv2212,info2201,info2202,info2205,info2208,inf
o2210,info2211,info2221,info2223,info2232,info2233]),
inscrit(tu,[info2112,sent2288]),
inscrit(turine,[ielv2113,info2101,info2102,info2105,info2106,info2107,inf
o2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(vaesen,[info2322,ielv2307,info2301,info2302,info2303,info2306,inf
o2326,info2327,info2328]),
inscrit(van_den_hove_d_ertsenryck,[info2322,ielv2307,info2301,info2302,in
fo2303,info2306,info2324,info2328,info3101]),
inscrit(van_tongelen,[ielv2113,info2101,info2102,info2105,info2106,info21
07,info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(vander_schelden,[info2229,ielv2212,info2201,info2202,info2205,inf
o2208,info2210,info2211,info2223,info2226,info2231,info2232,info2233]),
inscrit(vanderose,[ielv2113,info2101,info2102,info2105,info2106,info2107,
info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(vanderwhale,[ielv2113,info2101,info2102,info2105,info2106,info210
7,info2108,info2109,info2110,info2111,info2112,sent2288]),
```

```
inscrit(verenne,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2323,info2324,info2328]),
inscrit(vilz,[info2322,ielv2307,info2301,info2302,info2303,info2306,info2
324,info2325,info2328]),
inscrit(voogd,[info2322,ielv2307,info2301,info2302,info2303,info2306,info
2324,info2326,info2328]),
inscrit(wathelet,[info2229,ielv2212,info2201,info2202,info2205,info2208,i
nfo2210,info2211,info2224,info2226,info2232,info2233]),
inscrit(willame,[info2322,ielv2307,info2301,info2302,info2303,info2306,in
fo2325,info2327,info2328]),
inscrit(willemyns,[ielv2113,info2101,info2102,info2105,info2106,info2107,
info2108,info2109,info2110,info2111,info2112,sent2288]),
inscrit(zuyderhoff,[info2322,ielv2307,info2301,info2302,info2303,info2306
,info2325,info2327,info2328])
].
```

Fichier des dates (dates.txt)

```
[  date( 1 ,' 31-mai AM '),
   date( 2 ,' 31-mai PM '),
   date( 3 ,' 1-juin AM '),
   date( 4 ,' 1-juin PM '),
   date( 5 ,' 2-juin AM '),
   date( 6 ,' 2-juin PM '),
   date( 7 ,' 3-juin AM '),
   date( 8 ,' 3-juin PM '),
   date( 9 ,' 4-juin AM '),
   date( 10 ,' 4-juin PM '),
   date( 11 ,' 5-juin AM '),
   date( 12 ,' 5-juin PM '),
   date( 13 ,' 6-juin AM '),
   date( 14 ,' 6-juin PM '),
   date( 15 ,' 7-juin AM '),
   date( 16 ,' 7-juin PM '),
   date( 17 ,' 8-juin AM '),
   date( 18 ,' 8-juin PM '),
   date( 19 ,' 9-juin AM '),
   date( 20 ,' 9-juin PM '),
   date( 21 ,' 10-juin AM '),
   date( 22 ,' 10-juin PM '),
   date( 23 ,' 11-juin AM '),
   date( 24 ,' 11-juin PM '),
   date( 25 ,' 12-juin AM '),
   date( 26 ,' 12-juin PM '),
   date( 27 ,' 13-juin AM '),
   date( 28 ,' 13-juin PM '),
   date( 29 ,' 14-juin AM '),
   date( 30 ,' 14-juin PM '),
   date( 31 ,' 15-juin AM '),
```

```
    date( 32 ,' 15-juin PM '),
    date( 33 ,' 16-juin AM '),
    date( 34 ,' 16-juin PM '),
    date( 35 ,' 17-juin AM '),
    date( 36 ,' 17-juin PM '),
    date( 37 ,' 18-juin AM '),
    date( 38 ,' 18-juin PM '),
    date( 39 ,' 19-juin AM '),
    date( 40 ,' 19-juin PM '),
    date( 41 ,' 20-juin AM '),
    date( 42 ,' 20-juin PM '),
    date( 43 ,' 21-juin AM '),
    date( 44 ,' 21-juin PM '),
    date( 45 ,' 22-juin AM '),
    date( 46 ,' 22-juin PM '),
    date( 47 ,' 23-juin AM '),
    date( 48 ,' 23-juin PM '),
    date( 49 ,' 24-juin AM '),
    date( 50 ,' 24-juin PM '),
    date( 51 ,' 25-juin AM '),
    date( 52 ,' 25-juin PM '),
    date( 53 ,' 26-juin AM '),
    date( 54 ,' 26-juin PM '),
    date( 55 ,' 27-juin AM '),
    date( 56 ,' 27-juin PM ')
].
```

Fichier des congés (conges.txt)

```
[3,4,17,18,19,20,31,32,45,46].
```

Fichier des paramètres (param.txt)

```
[56,10,2].
```

Annexe 3

Listing du programme (version « Horaires individuels »)

```
/*
*************************************************************************
*************************************************************************

            PROGRAMME DE CREATION DES HORAIRES D'EXAMEN

            AUTEUR: Etienne Vandeput

            VERSION: 2.0
            (incluant une modelisation plus complete des types
             d'examens et quelques suggestions de J.-M. Jacquet)

            DATE: mai 2003

*************************************************************************
*************************************************************************


CONVENTION:
-------------------------------------------------------------------------

   Pour pouvoir traiter le probleme par la programmation par contraintes,
   les journees d'examens sont divisees en demi-jours numerotes a partir
   de 1.

   Ex: 31 mai am = 1 (debut de la session)
       31 mai pm = 2
       1 juin am = 3
       ...

   Les dimanches et jours feries et autres jours a enlever de la session
   sont egalement repris dans cette numerotation.


GLOSSAIRE:
-------------------------------------------------------------------------

   * Nom_etudiant

      le nom d'un étudiant (sans espace ni apostrophe et commençant par
      une lettre minuscule)

      Ex: dupont

   * Cours

      la dénomination du cours sous sa forme d'identifiant classique

      Ex: info2110
```

* Type

une expression qui designe toutes les epreuves d'un meme cours et la maniere eventuelle dont elles se succedent

On peut trouver des expressions semblables a:

oral(N) --------> oral (maximumu N etudiants par demi-jour)

ecrit ----------> ecrit

ecrit*oral -----> ecrit et oral le meme jour

ecrit+oral(N) --> ecrit et oral

* DJ

le demi-jour d'examen (une valeur comprise entre 1 et le double du nombre de jours de session)

* G

le n° du groupe dont fait partie l'étudiant pour l'épreuve concernée ou "ecrit" si c'est une epreuve ecrite

* Nom_prof

le nom du professeur sous forme d'un identifiant en trois lettres minuscules

Ex: jmj

* Liste_de_cours

une liste de cours

Ex: [info2110,info2112,ielv2113]

* Liste_de_demijours

une liste de valeurs autorisées pour les demi-jours

Ex: [1,2,8,32,33,34]

* Date

une date au format JJ/MM/AAAA AM

Ex: 05/06/2003 PM

DONNEES:

Elles sont rassemblées dans différents fichiers. Le contenu de chaque fichier est un atome Prolog unifiable a une liste d'elements. Les fichiers sont les suivants:

* Un fichier profs.txt donnant une liste d'elements, chacun associant a chaque professeur la liste des cours qu'il dispense. Un element typique de ce fichier a la forme

  enseigne(Nom_prof,Liste_de_cours)

  Ex: enseigne(jmj, [info1122, info2233])

* Un fichier epreuves.txt comprenant une liste d'elements, chacun donnant, par cours, le(s) type(s) d'examen(s) considere(s). La forme generale de ces elements est

  epreuve(Cours,Type).

  Ex: epreuve(info1122,ecrit+oral(N))

* Un fichier inscr.txt donnant une liste d'elements, chacun associant a un etudiant la liste des epreuves qu'il doit presenter. Un element typique de ce fichier a la forme

  inscrit(Nom_etudiant,Liste_de_cours)

  Ex: inscrit(dupont,[info1122,info2233])

* Un fichier indispo.txt donnant une liste d'elements, chacun associant a un professeur la liste des demi-jours ou il est indisponible. Un element typique de ce fichier a la forme

  indispo(Nom_prof,Liste_de_demijours)

  Ex: indispo(jmj,[1,2,14,18,...])

* Un fichier dates.txt donnant une liste d'elements, chacun associant a un entier, la demi-journee de la session correspondante. Un element typique a la forme

  date(n,Date) (n nombre entier strictement positif)

  Ex: date(1,'16/08/2002 AM')

* Un fichier conges.txt donnant la liste des demi-jours de conges pour lesquels aucun examen ne peut etre organise. Le contenu typique a la forme

  Liste_de_demijours

  Ex: [5, 6, 19, 20, ...].

* Un fichier parametres.txt donnant le nombre de demi-journees d'examen ainsi que le nombre maximal de groupes pour un oral. Le contenu typique a la forme

  [n1,n2] (n1 et n2 nombres entiers strictement positifs)

  Ex: [60, 20].

PRECONDITIONS:
---

* Les identifiants des éléments des listes contenues dans les fichiers sont les suivants:

  - profs.txt: Nom_prof (argument 1)
  - epreuves.txt: Cours (argument 1)
  - inscr.txt: Nom_etudiant (argument 1)
  - indispo.txt: Nom_prof (argument 1)
  - dates.txt: DJ (argument 1)

* Le nombre d'etudiants par demi-jour d'oral est strictement positif.

* Le nombre de demi-jours d'examen est strictement positif.

* Le nombre maximal de groupes pour un oral est strictement positif.

* La liste des conges est une sous-liste de la liste des valeurs correspondant aux demi-jours.

* La liste des indisponibilités pour chaque professeur est une sous-liste de la liste des valeurs correspondant aux demi-jours.

* Les dates reprises dans le fichier dates.txt sont valides.

SCHEMA E-R-A ET CARDINALITES:
---

* Les contraintes de cardinalite liees aux cours sont minimales.

Par exemple:

- un cours peut etre mentionne pour un etudiant dans le fichier inscr.txt et ne pas etre repris dans le fichier epreuves.txt

- un cours peut être mentionne pour un professeur dans le fichier profs.txt et ne pas etre repris dans le fichier epreuves.txt

- un cours peut exister sans qu'aucun etudiant n'y soit inscrit

RESULTATS:
---

* Dans l'etat actuel des choses, ils sont presentes dans un document au format HTML. L'horaire des epreuves pour l'ensemble des trois annees est presente dans un tableau en trois colonnes.

  Dans l'optique d'une solution individuelle, l'horaire personnel de chaque etudiant y figurera également.

```
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
O                                                                             O
O                          PROGRAMME PRINCIPAL                                O
O                                                                             O
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
```

PREDICAT:

---

horaire

STRUCTURE:

Sa structure generale est la suivante:

- construction de listes FDE et FDP d'elements contenant des references a des variables de domaine fini correspondant aux demi-jours d'examen pour les etudiants et les professeurs
- description des contraintes a appliquer a ces variables
- labeling des variables
- impression des resultats dans un fichier

Elle est reprise dans le module principal ci-dessous. Le prédicat "horaire" reussit si une instanciation de toutes les variables est possible.

CODE:

```
==========================================================================
*/

horaire:-

   write('\n\nCréation des listes de variables FD...\n\n'),

   listesFD(FDE,FDP),

   write('Prise en compte des contraintes...\n\n'),

   contraintes(FDE,FDP),

   write('Début de la labelisation...\n\n'),
```

```
/*
-------------------------------------------------------------------------

Pour la version "groupes anonymes", utiliser le predicat label_v1 qui
suit au lieu du predicat label_v2 de la version "horaires individuels".

   label_v1(FDP),

-------------------------------------------------------------------------
*/

   label_v2(FDE,FDP),

   write('Impressions dans un fichier...\n\n'),

   impressions(FDE,FDP),

   write('TERMINE').


/*
=========================================================================



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                        +
+                  MODULE DE GESTION DES VARIABLES FD                    +
+                                                                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++



PREDICAT:
-------------------------------------------------------------------------

   listesFD(FDE,FDP)

   Directionalites: <-,->


OBJET:
-------------------------------------------------------------------------

   Le module cree, a partir des donnees contenues dans les fichiers
   profs.txt, epreuves.txt et inscr.txt, les variables servant a la
   recherche.

   Deux variables sont creees:

     * pour chaque triplet (etudiant, cours, type)

       - la valeur du demi-jour d'examen

       - le n° du groupe auquel il appartiendra si c'est une epreuve
         orale en plusieurs groupes
```

Une variable est creee:

* pour chaque triplet (professeur, cours, groupe)

  - la valeur du demi-jour d'examen

    Ex: s'il y a une epreuve ecrite et 5 groupes a former pour l'oral, il y aura 6 groupes d'epreuves.

UTILISATION:

Ces variables sont stockees dans deux listes associatives FDE et FDP. L'association se fait de la maniere suivante:

* des listes Prolog sont creees sur bases du contenu des fichiers textes,

  - LP: liste des professeurs
  - LE: liste des etudiants
  - LC: liste des cours

  Pour rappel,

  - la liste LP contient des éléments du type:

    enseigne(Nom_prof,Liste_de_cours)

  - la liste LC contient des éléments du type:

    epreuve(Cours,Type)

  - la liste LE contient des éléments du type:

    inscrit(Nom_etudiant,Liste_de_cours)

* les listes FDE et FDP sont creees a partir de ces 3 listes

  - FDE: elements du type

    passe(Nom_etudiant,Cours,Mode,DJ)

    ou DJ est une variable FD et Mode peut prendre les valeurs suivantes:

    - oral (cas ou l'oral se deroule pour tous l'apres-midi suivant l'ecrit du matin)
    - ecrit
    - oral(G) (avec G variable FD designant le n° du groupe)

  Ex: Dans passe(dupont,info1122,oral(G),X), X designe le n° du demi-jour correspondant a l'interrogation de dupont (faisant partie du groupe G) a l'oral du cours d'info 1122. La determination de la valeur de G sera egalement confiee au programme.

```
            Dans passe(durand,info2222,ecrit,X), X designe le n° du
            demi-jour correspondant a l'interrogation ecrite de dupont
            pour le cours d'info 2222.

            Dans passe(durand,info2222,oral,X), X designe le n° du
            demi-jour correspondant a l'interrogation de dupont a
            l'oral commun du cours d'info 2222.


      - FDP: elements du type

             examine(Nom_prof,Cours,Mode,DJ)

             ou DJ est une variable FD et Mode a pour valeur "ecrit",
             "oral", ou "oral(G)" avec G representant un n° de groupe.

        Ex: dans examine(jmj,info2211,oral(5),X), X designe le n° du
            demi-jour correspondant a l'interrogation du groupe 5 pour
            l'examen d'info2211 par jmj


CODE:
=============================================================================
*/

listesFD(FDE,FDP):-

   cree_liste('profs63.txt',LP),

   cree_liste('epr63.txt',LC),

   cree_liste('inscr63.txt',LE),

   cree_FDE(LE,LC,FDE),

   cree_FDP(LP,LE,LC,FDP).

/*
=============================================================================

*** PROCEDURE cree_liste(<nom de fichier>,L) ***

OBJET:
-----

   Cette procedure permet de transformer un fichier texte en une
   veritable liste Prolog.

   Elle est utilisee pour la creation de diverses listes dont:

      * la liste des etudiants,

      * la liste des professeurs,

      * la liste des cours.

   Directionalites: <+,->
```

```
CODE:
================================================================================
*/

cree_liste(File,L):-

   open(File,read,S),

   read(S,L),

   close(S).

/*
================================================================================


*** PROCEDURE cree_FDE(LE,LC,FDE) ***

OBJET:
-----

   Le but final de cette procedure est de creer une liste qui associe
   les variables FD aux donnees concernant les etudiants.

   Le fichier contenant le nombre de demi-jours de session et le nombre
   maximum de groupes pour un oral, de meme que le fichier des demi-
   jours de conge sont transformes en listes Prolog.  Toutes ces
   informations sont transmises a la procedure "genere_FDE".

   Pour rappel, FDE contient des elements du type

      passe(Nom_etudiant,Cours,Mode,DJ)

   ou DJ est une variable FD et Mode peut prendre les valeurs suivantes:

      * oral (cas ou l'oral se deroule pour tous l'apres-midi suivant
              l'ecrit du matin)
      * ecrit
      * oral(G) (avec G variable FD designant le n° du groupe)

   Directionalites: <+,+,->


CODE:
================================================================================
*/

cree_FDE(LE,LC,FDE):-

   cree_liste('param63.txt',[NDJ,NGMax,_]),

   cree_liste('conges63.txt',LCong),
   genere_FDE(LE,LC,NDJ,NGMax,LCong,FDE).
```

```
/*
=============================================================================

*** PROCEDURE genere_FDE(LE,LC,NDJ,NGMax,LCong,FDE) ***

OBJET:
-----

   Cette procedure va creer la liste FDE sur base de toutes les
   informations necessaires.

   Des traitements particuliers sont a prevoir en fonction du type
   d'epreuves (nombre et succession).

   Ce premier parcours est recursif sur la liste des etudiants.

   LCE est la liste des cours auxquels l'etudiant est inscrit.  Cette
   liste est tiree de la liste des etudiants et ne correspond pas
   necessairement avec la liste des cours.

   Directionalites: <+,+,+,+,+,->


CODE:
=============================================================================
*/

genere_FDE([],_,_,_,_,[]):-

   !.

genere_FDE([inscrit(Nom,LCE)|Rinscrits],LC,NDJ,NGMax,LCong,FDE):-

   gen_etud(Nom,LCE,LC,NDJ,NGMax,LCong,R1),

   genere_FDE(Rinscrits,LC,NDJ,NGMax,LCong,R2),

   append(R1,R2,FDE).



/*
=============================================================================

*** PROCEDURE gen_etud(Nom,LCE,LC,NDJ,NGMax,LCong,L) ***

OBJET:
-----

   La procedure cree la partie de liste L de la liste FDE qui concerne
   le cours C auquel est inscrit l'etudiant Nom.

   Si le cours C de la liste LCE ne fait pas partie de la liste LC, il
   n'y a rien a faire, sinon, il faut examiner le type du cours.

   Directionalites: <+,+,+,+,+,+,->
```

```
CODE:
================================================================================
*/

gen_etud(_,[],_,_,_,_,[]):-

   !.


gen_etud(Nom,[C|Rcours],LC,NDJ,NGMax,LCong,FDE):-

   member(epreuve(C,Type),LC),

   !,

   gen_etud_ok(Nom,C,Type,NDJ,NGMax,LCong,R1),

   gen_etud(Nom,Rcours,LC,NDJ,NGMax,LCong,R2),

   append(R1,R2,FDE).


gen_etud(Nom,[_|Rcours],LC,NDJ,NGMax,LCong,FDE):-

   gen_etud(Nom,Rcours,LC,NDJ,NGMax,LCong,FDE).


/*
================================================================================

*** PROCEDURE gen_etud_ok(Nom,C,Type,NDJ,NGMax,LCong,L) ***

OBJET:
-----

   La procedure cree la partie de liste FDE qui concerne le cours C
   auquel est inscrit l'etudiant Nom apres avoir verifie que ce cours
   est bien dans la liste des cours.

   Cette procedure doit creer autant d'elements dans la liste FDE qu'il
   y a d'epreuves a passer par l'etudiant pour ce cours (2 au plus).
   Elle doit egalement introduire, au niveau des variables,
   les contraintes qui peuvent l'etre:

      * pas d'examen un jour de conge

      * l'ecrit precede l'oral

      * si l'ecrit et l'oral se deroulent le meme jour

        - l'ecrit est le matin (demi-jour impair)
        - l'oral est l'apres-midi (demi-jour suivant)

   Quatre cas sont a envisager:
```

```
      * un ecrit et un oral le meme jour
      * un ecrit et un oral
      * un oral
      * un ecrit

   Directionalites: <+,+,+,+,+,+,->


CODE:
================================================================================
*/

gen_etud_ok(Nom,C,ecrit*oral,NDJ,_,LCong,
                     [ passe(Nom,C,ecrit,X), passe(Nom,C,oral,Y) ]):-

   !,

   fd_domain(X,1,NDJ),

   fd_domain(Y,1,NDJ),

   impair(X,1,NDJ),

   Y#=#X+1,

   pas_conge(X,LCong),

   pas_conge(Y,LCong).


gen_etud_ok(Nom,C,ecrit,NDJ,_,LCong,
                     [ passe(Nom,C,ecrit,X) ]):-

   !,

   fd_domain(X,1,NDJ),

   pas_conge(X,LCong).


gen_etud_ok(Nom,C,ecrit+oral(_),NDJ,NGMax,LCong,
                     [ passe(Nom,C,ecrit,X), passe(Nom,C,oral(G),Y) ]):-

   !,

   fd_domain(X,1,NDJ),

   fd_domain(Y,1,NDJ),

   fd_domain(G,1,NGMax),

   X#<#Y,

   pas_conge(X,LCong),

   pas_conge(Y,LCong).
```

```
gen_etud_ok(Nom,C,oral(_),NDJ,NGMax,LCong,
                    [ passe(Nom,C,oral(G),X) ]):-

   !,

   fd_domain(X,1,NDJ),

   fd_domain(G,1,NGMax),

   pas_conge(X,LCong).


/*
=============================================================================

*** PROCEDURE pas_conge(X,L) ***

OBJET:
-----

   La procedure restreind le domaine des variables FD en tenant compte
   des jours de conge de la session.

   Directionalites: <-,+>


CODE:
=============================================================================
*/

pas_conge(_,[]):-

   !.


pas_conge(X,[Y|L]):-

   X#\=#Y,

   pas_conge(X,L).


/*
=============================================================================

*** PROCEDURE impair(X,VI,VF) ***

OBJET:
-----

   Un ecrit, lorsqu'il est suivi d'un oral le meme jour, doit se
   derouler le matin.  Sa variable FD correspondante doit donc etre
   impaire.

   La procedure a pour but d'obliger une variable entiere a
   prendre une valeur impaire dans un intervalle [VI, VF] ou VI et VF
   sont des nombres entiers representant les valeurs initiales et
   finales.
```

```
   Directionalites: <-,+,+>

PRECONDITIONS:
--------------

   La valeur de depart est un nombre entier impair.

   La valeur finale est un nombre entier.

CODE:
================================================================================
*/

impair(_,VI,VF):-

   VI+1>VF,

   !.


impair(X,VI,VF):-

   X #\=#VI+1,

   impair(X,VI+2,VF).

/*
================================================================================


*** PROCEDURE cree_FDP(LP,LE,LC,FDP) ***

OBJET:
------

   Le but final de cette procedure est de creer une liste qui associe
   les variables FD aux donnees concernant les etudiants.

   Le fichier contenant le nombre de demi-jours de session et le nombre
   maximum de groupes pour un oral, de meme que le fichier des demi-
   jours de conge sont transformes en listes Prolog.  Toutes ces
   informations sont transmises a la procedure "genere_FDP".

   Pour rappel, FDP contient des elements du type

      examine(Nom_prof,Cours,Mode,DJ)

   ou DJ est une variable FD et Mode peut prendre les valeurs suivantes:

      * oral (cas ou l'oral se deroule pour tous l'apres-midi
              suivant l'ecrit du matin)
      * ecrit
      * oral(G) (avec G variable FD designant le n° du groupe)
```

```
   Directionalites: <+,+,+,->

CODE:
================================================================================
*/

cree_FDP(LP,LE,LC,FDP):-

   cree_liste('param63.txt',[NDJ,NGMax,_]),

   cree_liste('conges63.txt',LCong),

   genere_FDP(LP,LE,LC,NDJ,NGMax,LCong,FDP).

/*
================================================================================

*** PROCEDURE genere_FDP(LP,LE,LC,FDP) ***

OBJET:
-----

   Cette procedure va creer la liste FDP sur base de toutes les
   informations necessaires.

   Des traitements particuliers sont a prevoir en fonction du type
   d'epreuves (nombre et succession).

   Ce premier parcours est recursif sur la liste des professeurs.

   LCP est la liste des cours que le professeur est cense enseigner.
   Cette liste est tiree de la liste des professeurs et ne correspond
   pas necessairement avec la liste des cours.

   Directionalites: <+,+,+,->

CODE:
================================================================================
*/

genere_FDP([],_,_,_,_,_,[]):-

   !.

genere_FDP([enseigne(Nom,LCP)|Rprofs],LE,LC,NDJ,NGMax,LCong,FDP):-

   gen_prof(Nom,LCP,LE,LC,NDJ,NGMax,LCong,R1),

   genere_FDP(Rprofs,LE,LC,NDJ,NGMax,LCong,R2),

   append(R1,R2,FDP).
```

```
/*
=============================================================================

*** PROCEDURE gen_prof(Nom,LCP,LE,LC,NDJ,NGMax,LCong,FDP) ***

OBJET:
-----

   La procedure cree la partie de liste L de la liste FDP qui concerne
   le cours C dispense par le professeur Nom.

   Si le cours C de la liste LCP ne fait pas partie de la liste LC, il
   n'y a rien a faire, sinon, il faut examiner le type du cours.

   Directionalites: <+,+,+,+,+,+,+,->

CODE:
=============================================================================
*/

gen_prof(_,[],_,_,_,_,_,[]):-

   !.


gen_prof(Nom,[C|Rcours],LE,LC,NDJ,NGMax,LCong,FDP):-

   member(epreuve(C,Type),LC),

   !,

   gen_prof_ok(Nom,C,Type,LE,LC,NDJ,NGMax,LCong,R1),

   gen_prof(Nom,Rcours,LE,LC,NDJ,NGMax,LCong,R2),

   append(R1,R2,FDP).


gen_prof(Nom,[_|Rcours],LE,LC,NDJ,NGMax,LCong,FDP):-

   gen_prof(Nom,Rcours,LE,LC,NDJ,NGMax,LCong,FDP).


 /*
=============================================================================

*** PROCEDURE gen_prof_ok(Nom,C,TYpe,LE,LC,NDJ,NGMax,LCong,L) ***

OBJET:
-----

   La procedure cree la partie de liste FDP qui concerne le cours C
   dispense par le professeur Nom apres avoir verifie que ce cours est
   bien dans la liste des cours.

   Cette procedure doit creer autant d'elements dans la liste FDP qu'il
   y a d'epreuves a faire passer par le professeur pour ce cours (au
   plus, n+1 ou n est le nombre de groupes a l'oral).  Elle doit
```

egalement introduire, au niveau des variables, les contraintes qui peuvent l'etre:

* pas d'examen un jour de conge

* l'ecrit precede l'oral

* si l'ecrit et l'oral se deroulent le meme jour

  - l'ecrit est le matin (demi-jour impair)
  - l'oral est l'apres-midi (demi-jour suivant)

Quatre cas sont a envisager:

* un ecrit et un oral le meme jour

* un ecrit et un oral

* un oral

* un ecrit

Directionalites: <+,+,+,+,+,+,+,+,->

CODE:

```
================================================================================
*/

gen_prof_ok(Nom,C,ecrit*oral,_,_,NDJ,_,LCong,
            [examine(Nom,C,ecrit,X),examine(Nom,C,oral,Y)]):-

   fd_domain(X,1,NDJ),

   pas_conge(X,LCong),

   impair(X,1,NDJ),

   fd_domain(Y,1,NDJ),

   pas_conge(Y,LCong),

   Y#=#X+1.


gen_prof_ok(Nom,C,ecrit+oral(N),LE,_,NDJ,_,LCong,
            [examine(Nom,C,ecrit,X)|FDP]):-

   fd_domain(X,1,NDJ),

   pas_conge(X,LCong),

   combien(C,LE,M),

   G is ceiling(M/N),

   gen_prof_ok_oralecrit(G,X,Nom,C,NDJ,LCong,FDP).
```

```
gen_prof_ok(Nom,C,oral(N),LE,_,NDJ,_,LCong,FDP):-

   combien(C,LE,M),

   G is ceiling(M/N),

   gen_prof_ok_oral(G,Nom,C,NDJ,LCong,FDP).


gen_prof_ok(Nom,C,ecrit,_,_,NDJ,_,LCong,
            [examine(Nom,C,ecrit,X)]):-

   fd_domain(X,1,NDJ),

   pas_conge(X,LCong).


 /*
=============================================================================

*** PROCEDURE combien(C,LE,N) ***

OBJET:
-----

   La procedure calcule du nombre d'etudiants N inscrits au cours C a
   partir de la liste LE.

   Directionalites: <+,+,->


CODE:
=============================================================================
*/

combien(_,[],0):-

   !.


combien(C,[inscrit(_,Lcours)|Rinscrits],N):-
     member(C,Lcours),
     !,
     N=N1+1,
     combien(C,Rinscrits,N1).


combien(C,[inscrit(_,_)|Rinscrits],N):-
     combien(C,Rinscrits,N).


 /*
=============================================================================

*** PROCEDURE gen_prof_ok_oralecrit(G,X,Nom,C,NDJ,LCong,FDP) ***
```

```
OBJET:
-----

   La procedure attribue une valeur a chaque groupe pour l'oral dans le
   cas ou un ecrit le precede et cree les variables FD correspondantes:

      Ex: examine(fbo,info2211,oral(4),X1)
          examine(fbo,info2211,oral(3),X2)
          ...

   Les variables Xi sont contraintes par les demi-jours de conge et,
   comme il existe une epreuve ecrite, par la condition que l'oral doit
   suivre l'ecrit.

   Directionalites: <-,-,+,+,+,+,->


CODE:
================================================================================
*/

gen_prof_ok_oralecrit(0,_,_,_,_,_,[]):-

   !.


gen_prof_ok_oralecrit(G,X,Nom,C,NDJ,LCong,
                      [examine(Nom,C,oral(G),Y)|RFDE]):-

   G1 is G-1,

   fd_domain(Y,1,NDJ),

   pas_conge(X,LCong),

   Y#>#X,

   gen_prof_ok_oralecrit(G1,X,Nom,C,NDJ,LCong,RFDE).


 /*
================================================================================

*** PROCEDURE gen_prof_ok_oral(G,Nom,C,NDJ,LCong,L) ***

OBJET:
-----

   La procedure attribue une valeur a chaque groupe pour l'oral dans le
   cas ou il n'y a pas d'epreuve ecrite et cree les variables FD
   correspondantes:

      Ex: examine(jmj,info2222,oral(4),X1)
          examine(jmj,info2222,oral(3),X2)
          ...

   Comme il n'existe pas d'epreuve ecrite, les variables Xi sont
   simplement contraintes par les demi-jours de conge.
```

```
   Directionalites: <-,+,+,+,+,->

CODE:
================================================================================
*/

gen_prof_ok_oral(0,_,_,_,_,[]):-

   !.


gen_prof_ok_oral(G,Nom,C,NDJ,LCong,[examine(Nom,C,oral(G),X)|RFDE]):-

   G1 is G-1,

   fd_domain(X,1,NDJ),

   pas_conge(X,LCong),

   gen_prof_ok_oral(G1,Nom,C,NDJ,LCong,RFDE).

/*
================================================================================


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                              +
+                  MODULE DE DESCRIPTION DES CONTRAINTES                       +
+                                                                              +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


PREDICAT:
--------------------------------------------------------------------------------

   contraintesFD(FDE,FDP)

   Directionalites: <-,->


OBJET:
--------------------------------------------------------------------------------

   Le module genere les differentes contraintes qui doivent agir sur les
   variables qui ont ete creees dans le module precedent et qui sont
   disponibles a partir de FDE et FDP.


UTILISATION:
--------------------------------------------------------------------------------

   Les contraintes prises en compte sont les suivantes:
```

```
   * Les professeurs sont indisponibles certains jours ou demi-jours.

   * Les groupes sont constitues d'un nombre egal d'etudiants (a une
     unite pres) et chaque etudiant est associe a un seul groupe pour
     une epreuve donnee.

     Cette contrainte n'est pas prise en compte dans la version "groupes
     anonymes".

   * Les etudiants passent l'ecrit d'un meme cours au meme moment.  Les
     etudiants d'un meme groupe d'oral sont egalement presents au meme
     moment.  Dans chaque cas, le professeur est present.

   * Il est bon que les differents examens d'un meme etudiant soient
     distants de quelques demi-jours (valeur lue dans le fichier
     param.txt).

   * Un professeur ne peut interroger deux groupes au meme moment.

   * Un examen oral ne peut avoir lieu en meme temps qu'un ecrit, qu'un
     oral commun ou qu'un oral ne comprenant qu'un seul groupe.

   Les listes necessaires sont reconstituees:

   * LI: liste des indisponibilites des professeurs

      - element typique: indispo(Nom_prof,Liste_de_demijours)

   * LC: liste des cours

      - element typique: epreuve(Cours,Type)

   * LE: liste des etudiants

      - element typique: inscrit(Nom_etudiant,Liste_de_cours)


CODE:
=============================================================================
*/

contraintes(FDE,FDP):-

   cree_liste('indis63.txt',LI),

   prof_dispo(FDP,LI),

   cree_liste('inscr63.txt',LE),

   cree_liste('epr63.txt',LC),

   attrib_groupes(FDE,LC,LE),

   prof_etud(FDE,FDP),

   cree_liste('param63.txt',[_,_,D]),
```

```
   dist_exam(FDE,D),

   pas2groupes(FDP).

/*
-------------------------------------------------------------------------

La contrainte correspondant au predicat pas_ecrit_oral n'a pas de sens
dans la version "horaires individuels" puisque chaque etudiant est pris
en compte separement.

pas_ecrit_oral(FDP,FDP).

-------------------------------------------------------------------------



=========================================================================

*** PROCEDURE prof_dispo(FDP,LI) ***

OBJET:
-----

   Les variables associees aux professeurs sont contraintes par la liste
   LI des demi-jours d'indisponibilite de chacun d'eux.

   La liste LI comprend des elements du type

      indispo(Nom_prof,Liste_de_demijours)

   La liste FDP est parcourue de maniere recursive.

   Directionalites: <-,+>


CODE:
=========================================================================
*/

prof_dispo([],_):-

   !.


prof_dispo([examine(Nom,_,_,X)|RFDP],LI):-

   member(indispo(Nom,Lindispo),LI),

   restrict(X,Lindispo),

   prof_dispo(RFDP,LI).


 /*
=========================================================================

*** PROCEDURE restrict(X,L) ***
```

```
OBJET:
-----

   Cette procedure enleve des valeurs possibles de la variable X, celles
   qui se trouvent dans la liste L.

   Directionalites: <-,+>


CODE:
================================================================================
*/

restrict(_,[]):-

   !.


restrict(X,[J|Rjours]):-

      X#\=#J,

      restrict(X,Rjours).


/*
================================================================================

*** PROCEDURE prof_etud(FDE,FDP) ***

OBJET:
-----

   Cette procedure assure que les professeurs sont presents a tous leurs
   examens avec les etudiants concernes.

   La procédure est récursive sur la liste FDP.

   Directionalites: <-,->


CODE:
================================================================================
*/

prof_etud(_,[]):-

   !.


prof_etud(FDE,[examine(_,C,ecrit,X)|RFDP]):-

   !,

   prof_etud_ens(FDE,examine(_,C,ecrit,X)),

   prof_etud(FDE,RFDP).
```

```
prof_etud(FDE,[examine(_,C,oral,X)|RFDP]):-
   !,
   prof_etud_ens(FDE,examine(_,C,oral,X)),
   prof_etud(FDE,RFDP).

prof_etud(FDE,[examine(_,C,oral(N),X)|RFDP]):-
   !,
   prof_etud_ens(FDE,examine(_,C,oral(N),X)),
   prof_etud(FDE,RFDP).

prof_etud(FDE,[examine(_,_,_,_)|RFDP]):-
   prof_etud(FDE,RFDP).

/*
================================================================================

*** PROCEDURE prof_etud_ens(FDE,examine(_,C,_,X)) ***


OBJET:
-----

   Cette procedure assure que les etudiants du cours C sont presents en
   même temps que le professeur du cours:

      * qu'il s'agisse d'un ecrit,

      * qu'il s'agisse d'un oral commun,

      * qu'il s'agisse d'un oral par groupes.

   La procédure est récursive sur la liste FDE.

   Directionalites: <-,->


CODE:
================================================================================
*/

prof_etud_ens([],_):-
   !.
```

```
prof_etud_ens([passe(_,C,ecrit,X1)|RFDE],examine(_,C,ecrit,X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,ecrit,X)).

prof_etud_ens([passe(_,_,_,_)|RFDE],examine(_,C,ecrit,X)):-
   !,
   prof_etud_ens(RFDE,examine(_,C,ecrit,X)).

prof_etud_ens([passe(_,C,oral,X1)|RFDE],examine(_,C,oral,X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,oral,X)).

prof_etud_ens([passe(_,C,oral(N),X1)|RFDE],examine(_,C,oral(N),X)):-
   !,
   X#=#X1,
   prof_etud_ens(RFDE,examine(_,C,oral(N),X)).

prof_etud_ens([passe(_,_,_,_)|RFDE],examine(_,C,T,X)):-
   !,
   prof_etud_ens(RFDE,examine(_,C,T,X)).

/*
==========================================================================

*** PROCEDURE dist_exam(FDE,D) ***

OBJET:
-----

   Cette procedure assure que deux examens consecutifs d'un meme
   etudiant sont distants d'au moins D demi-jours.  Cette contrainte
   n'est evidemment pas d'application lorsque l'ecrit est suivi d'un
   oral le meme jour (Type: ecrit*oral).

   La valeur de D a ete lue dans le fichier param.txt.

   La procedure est recursive sur la liste FDE.  Chaque element est
   examine par rapport a tous ses suivants (predicat 'distance').
```

```
   Directionalites: <-,+>

CODE:
================================================================================
*/

dist_exam([_],_):-

   !.


dist_exam([passe(Nom,C,Type,X)|RFDE],D):-

   distance(passe(Nom,C,Type,X),RFDE,D),

   dist_exam(RFDE,D).


/*
================================================================================

*** PROCEDURE distance(E,L,D) ***

OBJET:
-----

   Cette procedure impose a la variable X referencee dans l'element E du
   type passe(Nom,_,_,X) d'etre distante des autres variables associees
   aux elements de la liste L qui ont la forme passe(Nom,_,_,_) a
   l'exception des cas ou l'ecrit et l'oral ont lieu le meme jour.

   Directionalites: <-,-,+>


CODE:
================================================================================
*/

distance(_,[],_):-

   !.


distance(passe(Nom,C,ecrit,X),[passe(Nom,C,oral,_)|RFDE],D):-

   !,

   distance(passe(Nom,C,ecrit,X),RFDE,D).


distance(passe(Nom,C,oral,X),[passe(Nom,C,ecrit,_)|RFDE],D):-

   !,

   distance(passe(Nom,C,oral,X),RFDE,D).
```

```
distance(passe(Nom,C,T,X),[passe(Nom,_,_,Y)|RFDE],D):-
    !,
    dist(X,Y)#>=#D,
    distance(passe(Nom,C,T,X),RFDE,D).

distance(passe(Nom,C,T,X),[passe(_,_,_,_)|RFDE],D):-
    !,
    distance(passe(Nom,C,T,X),RFDE,D).

/*
=============================================================================

*** PROCEDURE pas2groupes(FDP) ***

OBJET:
-----

   Cette procedure garantit que le professeur n'interroge pas deux
   groupes distincts au meme moment.

   Elle est recursive sur la liste FDP.

   Dans cette liste, les elements correspondant a un meme enseignant se
   suivent, ce qui simplifie les verifications.

   Directionalites: <->

CODE:
=============================================================================
*/

pas2groupes([]):-
    !.

pas2groupes([examine(Nom,_,_,X)|RFDP]):-
    verifie(examine(Nom,_,_,X),RFDP),
    pas2groupes(RFDP).

/*
=============================================================================

*** PROCEDURE verifie(E,L) ***
```

```
OBJET:
-----

   Cette procedure assure que la variable associee a l'element E est la
   seule a prendre cette valeur parmi celles qui sont associees au meme
   professeur.

   En d'autres termes, tous les elements du type examine(Nom,_,_,X) ont
   des valeurs de X differentes pour une meme valeur de Nom.

   Comme les elements correspondant a un meme professeur se suivent, les
   verifications sont simplifiees.

   Directionalites: <-,->


CODE:
================================================================================
*/

verifie(_,[]):-

  !.


verifie(examine(Nom,_,_,X),[examine(Nom,_,_,Y)|RFDP]):-

   !,

   X#\=#Y,

   verifie(examine(Nom,_,_,X),RFDP).


verifie(examine(Nom,_,_,_),[examine(M,_,_,_)|_]):-

   Nom\=M,

   !.


/*
================================================================================

*** PROCEDURE pas_ecrit_oral(FDP,FDP) ***

OBJET:
-----

   La procedure 'pas_ecrit_oral' assure qu'une epreuve qui est soit un
   ecrit, soit un oral commun, soit l'oral d'un cours qui ne compte
   qu'un seul groupe, n'a pas lieu en meme temps qu'une autre epreuve.

   Directionalites: <+,+>
```

```
CODE:
================================================================================
*/

pas_ecrit_oral([],_):-

   !.


pas_ecrit_oral([E|RFDP],FDP):-

   !,

   pas_oral(E,RFDP,FDP),

   pas_ecrit_oral(RFDP,FDP).


/*
================================================================================

*** PROCEDURE pas_oral(E,FDP,FDP) ***

OBJET:
-----

   La procedure 'pas_oral' compare chaque element de la liste FDP avec
   ceux qui suivent.  Il n'y a pas de contrainte supplémentaire si
   l'element et son suivant sont des oraux comportant plusieurs groupes.

   Directionalites: <+,+,+>


CODE:
================================================================================
*/

pas_oral(_,[],_):-

   !.


pas_oral(examine(_,C,oral(M),X),[examine(_,_,oral(N),_)|RFDP],FDP):-

   M\=1,

   N\=1,

   !,

   pas_oral(examine(_,C,oral(M),X),RFDP,FDP).


pas_oral(examine(_,C,oral(1),X),[examine(_,_,oral(N),_)|RFDP],FDP):-

   N\=1,

   member(examine(_,C,oral(2),_),FDP),
```

```
    !,
    pas_oral(examine(_,C,oral(1),X),RFDP,FDP).

pas_oral(examine(_,C1,oral(1),X),[examine(_,C2,oral(1),_)|RFDP],FDP):-
   member(examine(_,C1,oral(2),_),FDP),
   member(examine(_,C2,oral(2),_),FDP),
   !,
   pas_oral(examine(_,C1,oral(1),X),RFDP,FDP).

pas_oral(examine(_,C1,oral(M),X),[examine(_,C2,oral(1),_)|RFDP],FDP):-
    M\=1,
    member(examine(_,C2,oral(2),_),FDP),
    !,
    pas_oral(examine(_,C1,oral(M),X),RFDP,FDP).

pas_oral(examine(_,C,T,X),[examine(_,_,_,Y)|RFDP],FDP):-
    X#\=#Y,
    pas_oral(examine(_,C,T,X),RFDP,FDP).

/*
=============================================================================

*** PROCEDURE attrib_groupes(FDP,LC,LE) ***

OBJET:
-----

    Cette procedure repartit les etudiants dans des groupes en fonction
    du nombre d'etudiants maximum interroges par demi-jour.

    Elle est recursive sur la liste des cours LC.  Les cours pour
    lesquels il n'y a pas d'oral sont negliges.

    Directionalites: <-,+,+>

CODE:
=============================================================================
*/

attrib_groupes(_,[],_):-
    !.
```

```
attrib_groupes(FDE,[epreuve(C,ecrit+oral(N))|Rcours],LE):-
    !,
    calcule(LE,N,C,NEG,NGMax,NG),
    attribue(C,NEG,NGMax,NG,0,FDE),
    attrib_groupes(FDE,Rcours,LE).

attrib_groupes(FDE,[epreuve(C,oral(N))|Rcours],LE):-
    !,
    calcule(LE,N,C,NEG,NGMax,NG),
    attribue(C,NEG,NGMax,NG,0,FDE),
    attrib_groupes(FDE,Rcours,LE).

attrib_groupes(FDE,[epreuve(_,_)|Rcours],LE):-
    attrib_groupes(FDE,Rcours,LE).

/*
=============================================================================

*** PROCEDURE calcule(LE,N,C,NEG,NGMax,NG) ***

OBJET:
-----

    Cette procedure calcule le nombre de groupes a constituer.  Les
    groupes comprendront NEG ou NEG-1 etudiants.  Le nombre de groupes
    de NEG etudiants est egalement calcule, c'est NGMax.

    S'il n'y a pas d'inscrits au cours, le nombre total de groupes et le
    nombre de groupe de NEG etudiants sont automatiquement de 0.

    Directionalites: <+,+,+,-,-,->


CODE:
=============================================================================
*/

calcule(LE,N,C,NEG,NGMax,NG):-
    combien(C,LE,NE1),
    NE is NE1,
    NG is ceiling(NE/N),
```

```
   NG\=0,
   !,
   NEG is ceiling(NE/NG),
   NGMax is NE-(NEG-1)*NG.

calcule(LE,N,C,NEG,NGMax,NG):-
   combien(C,LE,NE1),
   NE is NE1,
   NG is ceiling(NE/N),
   NG=0,
   !,
   NEG is 0,
   NGMax is 0.

/*
=============================================================================

*** PROCEDURE attribue(C,NEG,NGMax,CG,CE,FDE) ***

OBJET:
-----

   Cette procedure attribue un numero de groupe a tous les etudiants du
   cours C en commençant par le numero le plus eleve et en stoppant
   lorsque CG vaut 0.

   CE represente le nombre d'etudiants deja dans le groupe, ce qui
   permet de savoir quand il faut changer de numero de groupe.

   Directionalites: <+,+,+,+,+,->


CODE:
=============================================================================
*/

attribue(_,_,_,0,_,_):-
   !.

attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
   N is NEG-2,
```

```
    CE<N,
    CG>NGMax,
    !,
    CE1 is CE+1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG,CE1,RFDE).

attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-2,
    CE=N,
    CG>NGMax,
    !,
    CE1 is 0,
    CG1 is CG-1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG1,CE1,RFDE).

attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-1,
    CE<N,
    CG=<NGMax,
    !,
    CE1 is CE+1,
    G#=#CG,
    attribue(C,NEG,NGMax,CG,CE1,RFDE).

attribue(C,NEG,NGMax,CG,CE,[passe(_,C,oral(G),_)|RFDE]):-
    N is NEG-1,
    CE=N,
    CG=<NGMax,
    !,
```

```
   CG1 is CG-1,

   CE1 is 0,

   G#=#CG,

   attribue(C,NEG,NGMax,CG1,CE1,RFDE).


attribue(C,NEG,NGMax,CG,CE,[passe(_,_,_,_)|RFDE]):-

   attribue(C,NEG,NGMax,CG,CE,RFDE).

/*
=============================================================================
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                            +
+                          MODULE DE LABELISATION                            +
+                                                                            +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

PREDICAT:

```
   label_v1(FDP)

   Directionalites: <+,+>
```

OBJET:

```
   Le module a pour but d'attribuer des valeurs a certaines variables.
   Il s'agit de la labelisation correspondant a la version "groupes
   anonymes".  Les etudiants ne sont pas consideres individuellement.

   Ce sont d'abord les variables correspondant aux epreuves ecrites qui
   sont fixees puis celles qui correspondent aux oraux en commencant par
   ceux des professeurs dont les disponibilites sont les moindres.

   Pour les variables correspondant aux etudiants, il n'y aura donc pas
   de labelisation.
```

UTILISATION:

```
   Une procedure pour la labelisation des variables pour les
   epreuves ecrites:
```

```
      * label_v1_ecrit

   Une procedure pour la labelisation des variables pour les oraux sur
   base de la disponibilite croissante des professeurs:

      * label_v1_prof


CODE:
============================================================================
*/

label_v1(FDP):-

   label_v1_ecrit(FDP),

   label_v1_prof(FDP).


/*
============================================================================

*** PROCEDURE label_t_ecrit(FDP) ***

OBJET:
-----

   La procedure fixe les valeurs des variables correspondant aux examens
   ecrits.

   L'heuristique utilisee ici est celle du choix de la valeur minimum.

   Directionalites: <+>


CODE:
============================================================================
*/

label_v1_ecrit([]):-

   !.


label_v1_ecrit([examine(_,_,ecrit,X)|RFDP]):-

   !,

   fd_labeling(X,[value_method(min)]),

   label_v1_ecrit(RFDP).


label_v1_ecrit([examine(_,_,_,_)|RFDP]):-

   label_v1_ecrit(RFDP).
```

```
/*
============================================================================

*** PROCEDURE label_v1_prof(FDP) ***

OBJET:
-----

   La procedure fixe les valeurs des variables correspondant aux examens
   oraux.  Les elements envisages en premier sont ceux dont les
   variables possedent le moins de valeurs attribuables.

   Pour cela, la liste FDP est d'abord indexee puis triee avant
   labelisation.

   Directionalites: <+>


CODE:
============================================================================
*/

label_v1_prof(FDP):-

   size_prof_list(FDP,FDP1),

   keysort(FDP1,FDP2),

   label_v11_pr(FDP2).


/*
============================================================================

*** PROCEDURE size_prof_list(FDP,FDP1) ***

OBJET:
-----

   La procedure transforme la liste FDP en une liste indexee FDP1 en
   fonction du nombre de valeurs libres pour les variables (predicat
   'fd-size').

   Directionalites: <+,->



CODE:
============================================================================
*/

size_prof_list([],[]):-

   !.
```

```
size_prof_list([examine(Nom,C,G,X)|R],[(N,examine(Nom,C,G,X))|Rs]):-
   fd_size(X,N),
   size_prof_list(R,Rs).

size_prof_list([passe(Nom,C,G,X)|R],[(N,passe(Nom,C,G,X))|Rs]):-
   fd_size(X,N),
   size_prof_list(R,Rs).


/*
=============================================================================

*** PROCEDURE label_v11_pr(FDP1) ***

OBJET:
-----

   La procedure fixe les valeurs des variables en commencant donc par
   celles qui ont le moins de valeurs encore possibles.

   L'heuristique utilisee ici est celle du choix d'une valeur aleatoire
   parmi les valeurs restantes.

   Directionalites: <+,->



CODE:
=============================================================================
*/

label_v11_pr([]):-
   !.

label_v11_pr([(_,examine(_,_,oral(_),X))|RFDP]):-
   !,
   fd_labeling(X,[value_method(random)]),
   label_v11_pr(RFDP).

label_v11_pr([(_,examine(_,_,_,_))|RFDP]):-
   label_v11_pr(RFDP).
```

```
/*
=============================================================================

PREDICAT:
-----------------------------------------------------------------------------

   label_v2(FDE,FDP)

   Directionalites: <-,->

OBJET:
-----------------------------------------------------------------------------

   Il s'agit de la version 2 du module qui a pour but d'attribuer des
   valeurs a chacune des variables, y compris les variables
   individuelles des etudiants.

   Ce sont d'abord les variables correspondant aux professeurs qui sont
   fixées en commencant par ceux dont les disponibilites sont les
   moindres.

   Pour les variables correspondant aux etudiants, il n'y aura pas de
   strategie particuliere si ce n'est que les valeurs sont choisies de
   maniere aleatoire.

UTILISATION:
-----------------------------------------------------------------------------

   Une procedure pour la labelisation des variables pour les
   professeurs:

      * label_prof

   Une procedure pour la labelisation des variables pour les etudiants:

      * label_etud

CODE:
=============================================================================
*/

label_v2(FDE,FDP):-

   label_prof(FDP),

   label_etud(FDE).

/*
=============================================================================

*** PROCEDURE label_prof(FDP) ***
```

```
OBJET:
-----

   LABELISATION DES VARIABLES-PROFESSEURS

   Avant de fixer les valeurs, les professeurs sont classés par ordre
   décroissant d'indisponibilité.

   La liste FDP est transformee en une liste FDP1 dont les elements ont
   la forme (N,examine(Nom,C,G,X)) ou N represente le nombre de valeurs
   possibles pour X (predicat sizelist).

   Cette nouvelle liste est triee en FDP2 sur base de la cle qui est ce
   nombre de jours (predicat keysort).  Ce sont les valeurs de FDP2 qui
   seront labelisees (predicat label_pr).

   Directionalites: <->


CODE:
================================================================================
*/


label_prof(FDP):-

   size_p_list(FDP,FDP1),

   keysort(FDP1,FDP2),

   label_pr(FDP2).

/*
================================================================================

*** PROCEDURE size_p_list(FDP,FDP1) ***

OBJET:
-----

   Creer une liste correspond a la liste FDP

   A un element du type examine(Nom,C,G,X) de la liste FDP, correspond
   un element du type (N,examine(Nom,C,G,X)) dans la liste FDP1.

   Directionalites: <-,->


CODE:
================================================================================
*/

size_p_list([],[]):-

   !.
```

```
size_p_list([examine(Nom,C,G,X)|R],[(N,examine(Nom,C,G,X))|Rs]):-
   fd_size(X,N),
   size_p_list(R,Rs).

/*
=============================================================================

*** PROCEDURE label_pr(FDP2) ***

OBJET:
-----
   Fixation des valeurs des variables FD des elements de la liste FDP2
   Directionalites: <->

CODE:
=============================================================================
*/
label_pr([]):-
   !.

label_pr([(_,examine(_,_,_,X))|R]):-
   fd_labeling(X,[variable_method(first_fail),value_method(random)]),
   label_pr(R).

/*
=============================================================================

*** PROCEDURE label_etud(FDE) ***

OBJET:
-----
   LABELISATION DES VARIABLES-ETUDIANTS
   Les valeurs des variables FD presentes dans les elements de la liste
   FDE sont fixees (en utilisant une methode aleatoire pour leur choix)
   parmi les valeurs restantes.
   Directionalites: <->

CODE:
=============================================================================
*/
```

```
label_etud([]):-

   !.


label_etud([passe(_,_,_,X)|RFDE]):-

   fd_labeling(X,[variable_method(first_fail),value_method(random)]),

   label_etud(RFDE).


/*
=============================================================================

*** PROCEDURE nbr_gr(C,NGC) ***

OBJET:
-----

   La procedure calcule le nombre de groupes NGC pour le cours C afin de
   repartir les etudiants dans ces groupes.

   Directionalites: <+,->


CODE:
=============================================================================
*/

nbr_gr(C,NGC):-

   cree_liste('inscr63.txt',LE),

   cree_liste('epr63.txt',LC),

   combien(C,LE,M),

   et_accept(C,LC,N),

   NGC is ceiling(M/N).


/*
=============================================================================

*** PROCEDURE et_accept(C,LC,N) ***


OBJET:
-----

   La procedure recherche le nombre N d'etudiants acceptes a un oral du
   cours C.

   Directionalites: <+,+,->
```

```
CODE:
================================================================================
*/

et_accept(C,LC,N):-

   member(epreuve(C,oral(N)),LC),

   !.


et_accept(C,LC,N):-

   member(epreuve(C,ecrit+oral(N)),LC),

   !.

/*
================================================================================
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                              +
+                            MODULE D'IMPRESSION                               +
+                                                                              +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
PREDICAT:
--------------------------------------------------------------------------------

   impression(FDE,FDP)

   Directionalites: <+,+>


OBJET:
--------------------------------------------------------------------------------

   Le module a pour but de creer un document HTML dont l'interpretation
   par un navigateur fournira l'affichage de l'horaire pour les trois
   annees concernees.

   Le module permet momentanement d'enregistrer quelques informations de
   controle dans un fichier texte (les listes FDE et FDP, notamment).


UTILISATION:
--------------------------------------------------------------------------------

   Une procedure pour le controle

      * controle(FDE,FDP)
```

```
   Une procedure pour l'impression dans un fichier HTML

      * imprime(FDE,FDP)

CODE:
================================================================================
*/

impressions(FDE,FDP):-

   controle(FDE,FDP),

   imprime(FDE,FDP).


/*
================================================================================

*** PROCEDURE controle(FDE,FDP) ***

OBJET:
-----

   C'est une procedure utilitaire qui va enregistrer dans un fichier
   texte les listes FDE et FDP a raison d'un element par ligne.

   La procedure affiche(/2) est definie plus loin dans le module des
   procedures utilitaires.

   Directionalites: <+,+>


CODE:
================================================================================
*/

controle(FDE,FDP):-

   open('resultats.txt',write,S2),

   affiche(FDE,S2),

   affiche(FDP,S2),

   close(S2).


/*
================================================================================

*** imprime(FDE,FDP) ***

OBJET:
-----

   La procedure imprime(/2) va exploiter les informations disponibles
   dans les listes FDE et FDP pour creer un fichier nomme resultats.htm
```

```
   et un fichier hor_ind.htm qui soient interpretables par un navigateur
   Web.

   L'affichage par le navigateur produit pour resultats.htm, un tableau
   qui met en parallele les examens des trois annees d'etude dans l'ordre
   chronologique (une ligne = 1/2 jour).  Pour hor_ind.htm, on aura une
   succession de tableaux reprenant les horaires individuels des
   etudiants (n'apparaitront que les demi-jours ou l'etudiant doit etre
   interroge.

   Il est prevu que ce module permette un affichage des horaires
   personnels des etudiants, ce qui justifie que la liste FDE soit prise
   en compte.

   La procedure debut_tab(/1) cree l'entete, la procedure horaire(/3)
   cree le corps du tableau et la procedure fin_tab(/1) le termine
   correctement.

   Le fichier LD des correspondances entre les valeurs et les dates doit
   être regenere.

   Directionalites: <+,+>


CODE:
=============================================================================
*/

imprime(FDE,FDP):-

   cree_liste('dates63.txt',LD),

   open('resultats.htm',write,S3),

   debut_tab(S3),

   horaire(FDP,LD,S3),

   fin_tab(S3),

   close(S3),

   cree_liste('inscr63.txt',LE),

   open('hor_ind.htm',write,S4),

   hor_etud(FDE,LE,LD,S4),

   close(S4).


/*
=============================================================================

*** PROCEDURE debut_tab(S) ***
```

```
OBJET:
-----

   La procedure enregistre les codes HTML d'initialisation du tableau
   et l'envoie vers le stream S.

   Directionalites: <+>


CODE:
================================================================================
*/

debut_tab(S):-

   write(S,'<table width="400" border=1><tr><td align="center" '),

   write(S,'colspan="4">').


/*
================================================================================

*** PROCEDURE horaire(FDP,LD,S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S les donnees qui doivent
   composer le tableau.

   Directionalites: <+,+,+>


CODE:
================================================================================
*/


horaire(FDP,LD,S):-

   entete(S),

   affiche_result(LD,FDP,S).


/*
================================================================================

*** PROCEDURE entete(S) ***

OBJET:
-----

   La procedure cree la premiere ligne du tableau constituee des
   rubriques listees en-dessous d'elle et l'envoie vers le stream S.

   Directionalites: <+,+,+>
```

```
CODE:
================================================================================
*/

entete(S):-

   write(S,'HORAIRE DE LA SESSION'),

   changeligne(S),

   write(S,'Jours'),

   changecellule(S),

   write(S,'1ere M'),

   changecellule(S),

   write(S,'2eme M'),

   changecellule(S),

   write(S,'3eme M'),

   changeligne(S).


/*
================================================================================

*** PROCEDURE changeligne(S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les codes HTML
   necessaires au changement de ligne du tableau.

   Directionalites: <+>


CODE:
================================================================================
*/

changeligne(S):-

   write(S,'</td></tr><tr><td width="100" align="center">').



/*
================================================================================

*** PROCEDURE changecellule(S) ***
```

```
OBJET:
-----

   La procedure cree et envoie vers le stream S, les codes HTML
   necessaires au changement de cellule du tableau.

   Directionalites: <+>


CODE:
================================================================================
*/

changecellule(S):-

   write(S,'</td><td width="100" align="center">').


/*
================================================================================

*** PROCEDURE affiche_result(LD,FDP,S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les veritables donnees
   du tableau.  Elle est recursive sur la liste des dates des demi-jours
   d'examens.  A chaque fois, le demi-jour est ecrit ainsi que les
   examens qui ont lieu au cours de ce demi-jour.

   Directionalites: <+,+,+>


CODE:
================================================================================
*/

affiche_result([],_,_):-

   !.


affiche_result([date(DJ,Date)|Rdates],FDP,S):-

   ecrit_jour(Date,S),

   ecrit_exam(DJ,FDP,S),

   affiche_result(Rdates,FDP,S).


/*
================================================================================

*** PROCEDURE ecrit_jour(Date,S) ***
```

```
OBJET:
-----

   La procedure cree et envoie vers le stream S, les codes HTML
   necessaires a l'ecriture de la date correspondant au demi-jour.

   Directionalites: <+,+>


CODE:
================================================================================
*/

ecrit_jour(Date,S):-

   write(S,Date),

   changecellule(S).


/*
================================================================================

*** PROCEDURE ecrit_exam(DJ,FDP,S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les donnees et codes
   HTML necessaires a l'etablissement de la partie de ligne du tableau
   qui correspond aux trois colonnes des annees d'etude.

   Les cours de la 1ere annee correspondant a DJ sont recherches et
   ecrits.  Un changement de cellule s'ensuit ainsi que la recherche et
   l'ecriture des cours de la 2eme annee.  Celle-ci est suivie d'un
   dernier changement de cellule avec recherche et ecriture des cours de
   la 3eme annee suivie elle-meme d'un changement de ligne.

   Directionalites: <+,+,+>


CODE:
================================================================================
*/

ecrit_exam(DJ,FDP,S):-

   findall(C1:G1,cours_de_1(C1,G1,DJ,FDP),L1),

   affiche(L1,S),

   write(S,' '),

   changecellule(S),

   findall(C2:G2,cours_de_2(C2,G2,DJ,FDP),L2),

   affiche(L2,S),
```

```
   write(S,' '),

   changecellule(S),

   findall(C3:G3,cours_de_3(C3,G3,DJ,FDP),L3),

   affiche(L3,S),

   write(S,' '),

   changeligne(S).


/*
=============================================================================

*** PROCEDURE cours_de_1(C,G,DJ,FDP) ***

OBJET:
-----

   La procedure recherche tout cours C dont le label commence par info21
   et dont l'examen a lieu lors du demi-jour DJ tout en capturant
   l'information concernant le groupe ou le type (ecrit, oral commun).

   Le cours ielv2113 fait aussi partie du lot.

   Directionalites: <-,-,+,+>

CODE:
=============================================================================
*/

cours_de_1(C,G,DJ,FDP):-

   member(examine(_,C,G,DJ),FDP),

   C@<info22,

   C@>if.


cours_de_1(ielv2113,G,DJ,FDP):-

   member(examine(_,ielv2113,G,DJ),FDP).


/*
=============================================================================

*** PROCEDURE cours_de_2(C,G,DJ,FDP) ***

OBJET:
-----

   La procedure recherche tout cours C dont le label commence par info22
   et dont l'examen a lieu lors du demi-jour DJ tout en capturant
```

```
   l'information concernant le groupe ou le type (ecrit, oral commun).

   Les cours ielv2212 et sent2288 font aussi partie du lot.

   Directionalites: <-,-,+,+>

CODE:
================================================================================
*/

cours_de_2(C,G,DJ,FDP):-

   member(examine(_,C,G,DJ),FDP),

   C@<info23,

   C@>=info22.


cours_de_2(ielv2212,G,DJ,FDP):-

   member(examine(_,ielv2212,G,DJ),FDP).


cours_de_2(sent2288,G,DJ,FDP):-

   member(examine(_,sent2288,G,DJ),FDP).


/*
================================================================================

*** PROCEDURE cours_de_3(C,G,DJ,FDP) ***

OBJET:
-----

   La procedure recherche tout cours C dont le label commence par info23
   et dont l'examen a lieu lors du demi-jour DJ tout en capturant
   l'information concernant le groupe ou le type (ecrit, oral commun).

   Le cours ielv2307 fait aussi partie du lot.

   Directionalites: <-,-,+,+>


CODE:
================================================================================
*/

cours_de_3(C,G,DJ,FDP):-

   member(examine(_,C,G,DJ),FDP),

   C@>=info23,

   C@<info24.
```

```
cours_de_3(ielv2307,G,DJ,FDP):-

   member(examine(_,ielv2307,G,DJ),FDP).


/*
============================================================================

*** PROCEDURE fin_tab(S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les codes HTML
   necessaires a la cloture du tableau.

   Directionalites: <+>


CODE:
============================================================================
*/

fin_tab(S):-

   write(S,'</td></tr></table>').

/*
============================================================================

*** PROCEDURE hor_etud(FDE,LE,LD,S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les donnees du fichier
   HTML dont l'interpretation donnera une page Web contenant les
   tableaux des horaires personnels de chaque etudiant.  Chaque tableau
   commence par le nom de l'etudiant et est separe du suivant par une
   ligne horizontale.

   Elle est recursive sur la liste des etudiants.

   Directionalites: <+,+,+,+>


CODE:
============================================================================
*/

hor_etud(_,[],_,_):-

   !.
```

```
hor_etud(FDE,[inscrit(Nom,_)|Rinscrits],LD,S):-
   debut_tab(S),
   write(S,Nom),
   changeligne(S),
   result(FDE,Nom,LD,S),
   fin_tab(S),
   write(S,'<HR>'),
   hor_etud(FDE,Rinscrits,LD,S).

/*
=============================================================================

*** PROCEDURE result(FDE,Nom,LD,S) ***

OBJET:
-----

   La procedure cree et envoie vers le stream S, les donnees du fichier
   HTML correspondant a l'etudiant Nom.  Une liste est creee avec les
   epreuves de l'etudiant dont un element type est interro(X,C,G).

   Cette liste est ensuite triee avant enregistrement des informations
   dans le fichier.

   Directionalites: <+,+,+,+>

CODE:
=============================================================================
*/

result(_,[],_,_):-
   !.

result(FDE,Nom,LD,S):-
   findall(interro(X,C,G),member(passe(Nom,C,G,X),FDE),LX),
   sort(LX,LY),
   enregistre(LY,LD,S).


/*
=============================================================================

*** PROCEDURE enregistre(Liste,LD,S) ***
```

```
OBJET:
-----

   La procedure cree et envoie vers le stream S, les donnees du fichier
   et les codes HTML correspondant a l'etudiant Nom.

   Directionalites: <+,+,+>


CODE:
================================================================================
*/

enregistre([],_,_):-

   !.


enregistre([interro(X,C,G)|Rinterro],LD,S):-

   member(date(X,Date),LD),

   !,

   write(S,Date),

   changecellule(S),

   write(S,C),

   changecellule(S),

   write(S,G),

   changeligne(S),

   enregistre(Rinterro,LD,S).

/*
================================================================================




++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                              +
+                          PROCEDURES UTILITAIRES                              +
+                                                                              +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


OBJET:
--------------------------------------------------------------------------------

   Effectuer des tests et des controles d'affichage
```

```
   L est une liste et ses éléments sont affichés ligne par ligne:
      * a l'ecran,
      * dans un fichier texte,
      * dans un fichier texte au format HTML.

CODE:
================================================================================
*/
affiche([]):-
   !.

affiche([X|L]):-
   write(X),
   write('\n'),
   affiche(L).


affiche([],_):-
   !.

affiche([X|L],S):-
     write(S,X),
     write(S,'<BR>'),
     affiche(L,S).
```

Annexe 4

Version « Groupes anonymes »

Trois voies sont suivies en fonction des choix d'heuristiques (voir chapitre 4)

Session de juin 2003 (version 1: choix aléatoire des valeurs)

Nous fournissons trois solutions.

Solution 1

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM	info2109:oral(2)		
31-mai PM		info2233:oral(4)	info2327:oral(1)
1-juin AM			
1-juin PM			
2-juin AM			info2325:oral(1)
2-juin PM	info2114:oral(1)		
3-juin AM		info2208:oral(5)	
3-juin PM		info2231:oral(1)	
4-juin AM	ielv2113:oral(1)		
4-juin PM		info2232:ecrit	
5-juin AM	info2109:oral(3)	info2208:oral(4)	info2324:oral(2)
5-juin PM		info2233:oral(6)	info2322:oral(2)
6-juin AM	info2102:oral(1)		
6-juin PM		info2204:ecrit	
7-juin AM	info2110:oral(1)		
7-juin PM		info2205:oral(1)	
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM		info2230:oral(1)	
10-juin PM		info2223:oral(1)	
11-juin AM		info2211:ecrit	
11-juin PM		ielv2212:oral(1)	
12-juin AM		info2222:ecrit	
12-juin PM		info2221:oral(1)	
13-juin AM	info2106:ecrit		
13-juin PM		info2233:oral(5)	
14-juin AM			info2326:oral(1)
14-juin PM	info2109:oral(4)		
15-juin AM			
15-juin PM		info2211:oral(1)	
16-juin AM		info2202:ecrit	
16-juin PM		info2202:oral	

HORAIRE DE LA SESSION			
17-juin AM			info2301:ecrit
17-juin PM	info2112:oral(1)		
18-juin AM		info2233:oral(3)	
18-juin PM	info2105:ecrit		
19-juin AM	info2105:oral(2)		info2301:oral(2)
19-juin PM	info2109:oral(1)	info2208:oral(2)	info2301:oral(1)
20-juin AM			ielv2307:oral(1)
20-juin PM	info2105:oral(4)		info2322:oral(1)
21-juin AM		info2233:oral(2)	
21-juin PM	info2111:ecrit		
22-juin AM			
22-juin PM			info2301:oral(4)
23-juin AM	info2105:oral(3)		info2324:oral(1)
23-juin PM	info2105:oral(1)	info2233:oral(1)	
24-juin AM			info2302:oral(1)
24-juin PM	info2108:ecrit		
25-juin AM		info2208:oral(1)	
25-juin PM		sent2288:ecrit	
26-juin AM	info2111:oral(2)	info2208:oral(3)	info2327:oral(2)
26-juin PM		info2211:oral(2	
27-juin AM		info2209:ecrit	
27-juin PM			info2327:oral(3)
28-juin AM	info2111:oral(3)		info2322:oral(3) info2301:oral(3)
28-juin PM		info2211:oral(3)	
29-juin AM			
29-juin PM	info2105:oral(5) info2111:oral(1)		
30-juin AM	info2111:oral(4)		info2301:oral(5) info2326:oral(2)
30-juin PM	info2107:ecrit		

Solution 2

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		ielv2212:oral(1)	
31-mai PM		info2233:oral(4)	
1-juin AM			
1-juin PM			
2-juin AM			info2327:oral(1)
2-juin PM		info2204:ecrit	
3-juin AM	ielv2113:oral(1)		
3-juin PM	info2109:oral(2)		
4-juin AM		info2230:oral(1)	
4-juin PM		info2208:oral(1)	info2324:oral(2)

HORAIRE DE LA SESSION			
5-juin AM		info2223:oral(1)	
5-juin PM			
6-juin AM		info2209:ecrit	
6-juin PM		info2208:oral(5)	
7-juin AM			
7-juin PM	info2110:oral(1)		
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM			info2325:oral(1)
10-juin PM	info2102:oral(1)		
11-juin AM		info2231:oral(1)	
11-juin PM		info2232:ecrit	
12-juin AM	info2105:ecrit		
12-juin PM		info2222:ecrit	
13-juin AM			
13-juin PM	info2114:oral(1)		
14-juin AM			
14-juin PM	info2108:ecrit		
15-juin AM	info2105:oral(1)		
15-juin PM			
16-juin AM	info2105:oral(5)	info2208:oral(4)	
16-juin PM		info2221:oral(1)	
17-juin AM		info2202:ecrit	
17-juin PM		info2202:oral	
18-juin AM	info2111:ecrit		
18-juin PM		info2205:oral(1)	
19-juin AM	info2109:oral(3)	info2208:oral(2)	
19-juin PM	info2111:oral(2)	info2233:oral(2)	info2322:oral(2) info2327:oral(3)
20-juin AM		info2233:oral(5) info2208:oral(3)	
20-juin PM			info2322:oral(3)
21-juin AM			info2301:ecrit
21-juin PM	info2105:oral(3) info2111:oral(1)	info2233:oral(3)	
22-juin AM			
22-juin PM			info2301:oral(1)
23-juin AM			info2327:oral(2)
23-juin PM	info2105:oral(2)		info2301:oral(4) info2324:oral(1)
24-juin AM	info2109:oral(1) info2111:oral(4)		info2301:oral(5) info2326:oral(1)
24-juin PM			info2302:oral(1)

HORAIRE DE LA SESSION			
25-juin AM			info2301:oral(3)
25-juin PM		sent2288:ecrit	
26-juin AM	info2109:oral(4)		info2322:oral(1) info2301:oral(2) info2326:oral(2)
26-juin PM		info2233:oral(1)	
27-juin AM	info2112:oral(1)		
27-juin PM			ielv2307:oral(1)
28-juin AM	info2106:ecrit		
28-juin PM		info2211:ecrit	
29-juin AM	info2111:oral(3)	info2211:oral(2)	
29-juin PM	info2105:oral(4)	info2211:oral(3)	
30-juin AM		info2233:oral(6) info2211:oral(1)	
30-juin PM	info2107:ecrit		

Solution 3

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		info2233:oral(4)	
31-mai PM			
1-juin AM			
1-juin PM			
2-juin AM			ielv2307:oral(1)
2-juin PM		info2211:ecrit	
3-juin AM			info2301:ecrit
3-juin PM		info2233:oral(6)	
4-juin AM		info2208:oral(1)	info2327:oral(1)
4-juin PM		ielv2212:oral(1)	
5-juin AM			info2322:oral(3) info2301:oral(2)
5-juin PM		info2208:oral(3)	
6-juin AM		info2230:oral(1)	
6-juin PM		info2211:oral(3) info2208:oral(4)	
7-juin AM	info2109:oral(3)	info2233:oral(3)	info2324:oral(2)
7-juin PM			
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2107:ecrit		
10-juin PM		info2232:ecrit	
11-juin AM	info2102:oral(1)		
11-juin PM			info2327:oral(3) info2301:oral(1)

HORAIRE DE LA SESSION			
12-juin AM	info2114:oral(1)		
12-juin PM			
13-juin AM	info2109:oral(2)	info2208:oral(2)	info2327:oral(2)
13-juin PM		info2204:ecrit	
14-juin AM	info2111:ecrit		
14-juin PM		info2233:oral(1)	info2324:oral(1)
15-juin AM	info2111:oral(2)		
15-juin PM	info2111:oral(1)		
16-juin AM		info2202:ecrit	
16-juin PM		info2202:oral	
17-juin AM	info2105:ecrit		
17-juin PM			info2325:oral(1)
18-juin AM		info2222:ecrit	
18-juin PM	info2108:ecrit		
19-juin AM		info2231:oral(1)	
19-juin PM		info2221:oral(1)	
20-juin AM	info2106:ecrit		
20-juin PM		info2223:oral(1)	
21-juin AM	info2105:oral(4)	info2233:oral(2)	
21-juin PM			info2301:oral(5) info2326:oral(1)
22-juin AM		info2211:oral(2)	
22-juin PM			
23-juin AM	info2109:oral(4)		
23-juin PM	ielv2113:oral(1)		
24-juin AM			info2302:oral(1)
24-juin PM	info2111:oral(4)	info2233:oral(5)	info2322:oral(1)
25-juin AM		sent2288:ecrit	
25-juin PM	info2110:oral(1)		
26-juin AM	info2109:oral(1)		info2301:oral(3)
26-juin PM		info2209:ecrit	
27-juin AM	info2112:oral(1)		
27-juin PM	info2105:oral(3)		
28-juin AM		info2205:oral(1)	
28-juin PM			info2326:oral(2)
29-juin AM	info2105:oral(2) info2111:oral(3)		
29-juin PM			
30-juin AM	info2105:oral(5)	info2211:oral(1) info2208:oral(5)	
30-juin PM	info2105:oral(1)		info2322:oral(2) info2301:oral(4)

Session de juin 2003 (version 2: choix de la plus petite des valeurs)

Nous fournissons deux solutions.

Solution 1

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		info2211:ecrit	
31-mai PM			info2301:ecrit
1-juin AM			
1-juin PM			
2-juin AM	info2106:ecrit		
2-juin PM		info2209:ecrit	
3-juin AM		info2222:ecrit	
3-juin PM	info2107:ecrit		
4-juin AM		info2204:ecrit	
4-juin PM	info2111:ecrit		
5-juin AM		info2232:ecrit	
5-juin PM			
6-juin AM			
6-juin PM			
7-juin AM			
7-juin PM			
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2105:ecrit		
10-juin PM			
11-juin AM		info2202:ecrit	
11-juin PM		info2202:oral	
12-juin AM		info2205:oral(1)	
12-juin PM		ielv2212:oral(1)	
13-juin AM		info2230:oral(1)	
13-juin PM		info2223:oral(1)	
14-juin AM	info2108:ecrit		
14-juin PM		info2231:oral(1)	
15-juin AM			
15-juin PM			
16-juin AM	info2114:oral(1)		
16-juin PM			info2325:oral(1)
17-juin AM		info2208:oral(5)	info2301:oral(4)
17-juin PM	info2111:oral(4)	info2233:oral(6) info2208:oral(3)	info2301:oral(2)
18-juin AM	info2109:oral(4) info2111:oral(2)	info2233:oral(4) info2211:oral(3) info2208:oral(1)	
18-juin PM	info2102:oral(1)		

HORAIRE DE LA SESSION			
19-juin AM	info2109:oral(2) info2105:oral(5) info2111:oral(1)	info2233:oral(2) info2211:oral(1) info2208:oral(2)	info2322:oral(2) info2327:oral(2)
19-juin PM	info2109:oral(1) info2105:oral(3) info2111:oral(3)	info2233:oral(1) info2211:oral(2) info2208:oral(4)	info2322:oral(1) info2327:oral(1) info2324:oral(1)
20-juin AM	info2109:oral(3) info2105:oral(1)	info2233:oral(3)	info2322:oral(3) info2327:oral(3) info2301:oral(1) info2324:oral(2)
20-juin PM		info2221:oral(1)	
21-juin AM	info2105:oral(2)	info2233:oral(5)	info2301:oral(3)
21-juin PM	info2105:oral(4)		info2301:oral(5) info2326:oral(2)
22-juin AM			
22-juin PM			
23-juin AM			info2326:oral(1)
23-juin PM	info2112:oral(1)		
24-juin AM			info2302:oral(1)
24-juin PM	info2110:oral(1)		
25-juin AM		sent2288:ecrit	
25-juin PM	ielv2113:oral(1)		
26-juin AM			ielv2307:oral(1)
26-juin PM			
27-juin AM			
27-juin PM			
28-juin AM			
28-juin PM			
29-juin AM			
29-juin PM			
30-juin AM			
30-juin PM			

Solution 2

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		info2211:ecrit	
31-mai PM	info2111:ecrit		
1-juin AM			
1-juin PM			
2-juin AM			info2301:ecrit
2-juin PM			
3-juin AM		info2222:ecrit	
3-juin PM	info2106:ecrit		
4-juin AM			
4-juin PM			

HORAIRE DE LA SESSION			
5-juin AM		info2209:ecrit	
5-juin PM			
6-juin AM			
6-juin PM	info2107:ecrit		
7-juin AM			
7-juin PM		info2205:oral(1)	
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2105:ecrit		
10-juin PM		ielv2212:oral(1)	
11-juin AM		info2230:oral(1)	
11-juin PM			info2302:oral(1)
12-juin AM		info2202:ecrit	
12-juin PM		info2202:oral	
13-juin AM		info2223:oral(1)	
13-juin PM			info2325:oral(1)
14-juin AM		info2204:ecrit	
14-juin PM	info2114:oral(1)		
15-juin AM			
15-juin PM	info2111:oral(4)	info2211:oral(3)	
16-juin AM	info2108:ecrit		
16-juin PM		info2211:oral(1) info2208:oral(5)	
17-juin AM	info2111:oral(2)	info2233:oral(6) info2211:oral(2) info2208:oral(3)	
17-juin PM		info2232:ecrit	
18-juin AM	info2109:oral(4) info2111:oral(1)	info2233:oral(4) info2208:oral(1)	info2301:oral(4)
18-juin PM	info2109:oral(2) info2111:oral(3)	info2233:oral(2) info2208:oral(2)	info2327:oral(2) info2301:oral(2)
19-juin AM	info2102:oral(1)		
19-juin PM		info2221:oral(1)	
20-juin AM	info2109:oral(1) info2105:oral(5)	info2233:oral(1) info2208:oral(4)	info2322:oral(3) info2327:oral(1) info2301:oral(1) info2324:oral(1)
20-juin PM	info2109:oral(3) info2105:oral(3)	info2233:oral(3)	info2322:oral(1) info2327:oral(3) info2301:oral(3) info2324:oral(2)
21-juin AM		info2231:oral(1)	
21-juin PM	info2105:oral(1)	info2233:oral(5)	info2301:oral(5)
22-juin AM	info2105:oral(2)		
22-juin PM	info2105:oral(4)		

HORAIRE DE LA SESSION			
23-juin AM	info2112:oral(1)		
23-juin PM	info2110:oral(1)		
24-juin AM			info2322:oral(2) info2326:oral(1)
24-juin PM			info2326:oral(2)
25-juin AM		sent2288:ecrit	
25-juin PM	ielv2113:oral(1)		
26-juin AM			ielv2307:oral(1)
26-juin PM			
27-juin AM			
27-juin PM			
28-juin AM			
28-juin PM			
29-juin AM			
29-juin PM			
30-juin AM			
30-juin PM			

Session de juin 2003 (version 1: choix de la plus petite seulement pour les écrits et aléatoire pour les autres)

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		info2211:ecrit	
31-mai PM	info2111:ecrit		
1-juin AM			
1-juin PM			
2-juin AM			info2301:ecrit
2-juin PM			info2325:oral(1)
3-juin AM		info2222:ecrit	
3-juin PM	info2106:ecrit		
4-juin AM	info2109:oral(1)	info2233:oral(5)	info2301:oral(2)
4-juin PM		info2223:oral(1)	
5-juin AM		info2209:ecrit	
5-juin PM	info2111:oral(2)	info2211:oral(1)	info2322:oral(1)
6-juin AM		info2233:oral(2)	
6-juin PM	info2107:ecrit		
7-juin AM	ielv2113:oral(1)		
7-juin PM			
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2105:ecrit		
10-juin PM		info2208:oral(5)	info2322:oral(2)

HORAIRE DE LA SESSION			
11-juin AM			info2302:oral(1)
11-juin PM		info2233:oral(3)	
12-juin AM		info2202:ecrit	
12-juin PM		info2202:oral	
13-juin AM			
13-juin PM		info2208:oral(4)	
14-juin AM		info2204:ecrit	
14-juin PM			
15-juin AM	info2105:oral(5)		
15-juin PM	info2105:oral(3)		
16-juin AM	info2108:ecrit		
16-juin PM		info2233:oral(4)	
17-juin AM			info2327:oral(3)
17-juin PM		info2232:ecrit	
18-juin AM		info2211:oral(2)	info2324:oral(2)
18-juin PM	info2105:oral(2)	info2233:oral(6) info2208:oral(1)	
19-juin AM			ielv2307:oral(1)
19-juin PM	info2111:oral(1)	info2233:oral(1)	
20-juin AM	info2109:oral(3)		info2322:oral(3)
20-juin PM		info2208:oral(3)	info2301:oral(4)
21-juin AM		info2231:oral(1)	
21-juin PM	info2111:oral(3)		info2301:oral(5) info2326:oral(2)
22-juin AM	info2105:oral(4) info2111:oral(4)		
22-juin PM			
23-juin AM	info2112:oral(1)		
23-juin PM			info2327:oral(1) info2301:oral(1) info2324:oral(1)
24-juin AM		info2205:oral(1)	
24-juin PM	info2109:oral(4)		
25-juin AM		sent2288:ecrit	
25-juin PM	info2110:oral(1)		
26-juin AM		ielv2212:oral(1)	
26-juin PM	info2114:oral(1)		
27-juin AM		info2221:oral(1)	
27-juin PM		info2208:oral(2)	
28-juin AM		info2230:oral(1)	
28-juin PM	info2102:oral(1)		
29-juin AM			info2301:oral(3)
29-juin PM			
30-juin AM	info2109:oral(2) info2105:oral(1)	info2211:oral(3)	info2327:oral(2) info2326:oral(1)
30-juin PM			

Version « Horaires individuels »

Session de juin 2003 (données corrigées)

Cet horaire a été calculé à un moment où l'horaire réalisé manuellement avait été communiqué. Nous avons pu disposer des données complètes si ce n'est que certains étudiants étaient inscrits à trop d'épreuves. Nous avons apporté les corrections nécessaires. Les résultats sans ces corrections suivent ceux-ci.

HORAIRE DE LA SESSION			
Jours	1ère M	2e M	3e M
31-mai AM		info2211:ecrit	
31-mai PM			info2327:oral(3)
1-juin AM			
1-juin PM			
2-juin AM	info2109:oral(1) info2112:oral(4	info2226:ecrit	info2325:oral(1)
2-juin PM			
3-juin AM	info2102:oral(4)	info2201:ecrit	
3-juin PM	info2110:oral(1)		
4-juin AM	info2102:oral(3)	info2211:oral(1) info2204:ecrit	
4-juin PM		info2208:oral(4)	info2324:oral(1)
5-juin AM			
5-juin PM	ielv2113:oral(1)	info2232:ecrit	
6-juin AM			
6-juin PM	info2101:ecrit		info2304:oral(1)
7-juin AM		info2205:oral(1)	
7-juin PM	info2107:ecrit		
8-juin AM			
8-juin PM			
9-juin AM			
9-juin PM			
10-juin AM	info2112:oral(2)	info2209:ecrit info2210:oral(1)	
10-juin PM	info2112:oral(3)		info2322:oral(4)
11-juin AM		info2223:oral(1) info2230:oral(2)	
11-juin PM	info2109:oral(4) info2102:oral(2)	info2230:oral(3)	info2302:oral(1)
12-juin AM		info2211:oral(4)	
12-juin PM		info2230:oral(1) info2208:oral(3)	info2301:ecrit
13-juin AM	info2105:ecrit		
13-juin PM		info2229:ecrit	ielv2307:oral(1)
14-juin AM	info2109:oral(3)		
14-juin PM		info2231:oral(1) info2221:oral(1)	info2327:oral(2)
15-juin AM			

HORAIRE DE LA SESSION			
15-juin PM			
16-juin AM	info2110:oral(2)	info2230:oral(5)	info2326:oral(1)
16-juin PM	info2112:oral(1)	info2208:oral(1)	
17-juin AM	info2114:oral(1)	info2211:oral(3) info2230:oral(4) info2208:oral(2)	
17-juin PM			info2306:ecrit
18-juin AM	info2111:ecrit		
18-juin PM		info2208:oral(5)	info2324:oral(2)
19-juin AM	info2105:oral(1) info2111:oral(2)	info2233:oral(1)	info2322:oral(1)
19-juin PM	info2111:oral(3)		
20-juin AM	info2105:oral(2)		info2322:oral(3) info2301:oral(5)
20-juin PM	info2105:oral(3)	info2233:oral(5) info2211:oral(2)	
21-juin AM	info2105:oral(4)		
21-juin PM	info2105:oral(5)	info2233:oral(4)	info2301:oral(1) info2324:oral(3)
22-juin AM			
22-juin PM			
23-juin AM	info2106:ecrit	info2233:oral(3)	info2301:oral(2)
23-juin PM			
24-juin AM	info2109:oral(2) info2111:oral(1)	info2233:oral(2)	
24-juin PM			info2322:oral(2)
25-juin AM		info2233:oral(6) info2222:ecrit sent2288:ecrit	
25-juin PM			info2301:oral(4)
26-juin AM	info2108:ecrit	ielv2212:oral(1)	
26-juin PM			info2326:oral(2)
27-juin AM	info2110:oral(3)	info2202:ecrit	
27-juin PM	info2102:oral(1)	info2202:oral	info2327:oral(1) info2301:oral(3)

Voici quelques horaires individuels (notamment les premiers et les derniers pour ne pas laisser supposer une sélection partiale).

achbany		
31-mai AM	info2211	ecrit
2-juin AM	info2226	ecrit
3-juin AM	info2201	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
12-juin AM	info2211	oral(4)
13-juin PM	info2229	ecrit

achbany		
16-juin AM	info2230	oral(5)
18-juin PM	info2208	oral(5)
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

anciaux		
31-mai AM	info2211	ecrit
3-juin AM	info2201	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
12-juin AM	info2211	oral(4)
14-juin PM	info2221	oral(1)
16-juin AM	info2230	oral(5)
18-juin PM	info2208	oral(5)
20-juin AM	info2224	ecrit
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

andre		
2-juin AM	info2112	oral(4)
3-juin AM	info2102	oral(4)
5-juin PM	ielv2113	oral(1)
6-juin PM	info2101	ecrit
7-juin PM	info2107	ecrit
11-juin PM	info2109	oral(4)
13-juin AM	info2105	ecrit
18-juin AM	info2111	ecrit
19-juin PM	info2111	oral(3)
21-juin PM	info2105	oral(5)
23-juin AM	info2106	ecrit
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit
27-juin AM	info2110	oral(3)

arman		
31-mai AM	info2211	ecrit
3-juin AM	info2201	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)

arman		
10-juin AM	info2210	oral(1)
12-juin AM	info2211	oral(4)
13-juin PM	info2229	ecrit
16-juin AM	info2230	oral(5)
17-juin AM	info2224	ecrit
18-juin PM	info2208	oral(5)
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

barras		
31-mai AM	info2211	ecrit
3-juin AM	info2201	ecrit
4-juin PM	info2224	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
11-juin AM	info2223	oral(1)
12-juin AM	info2211	oral(4)
16-juin AM	info2230	oral(5)
18-juin PM	info2208	oral(5)
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

...

hallez		
4-juin PM	info2323	oral(1)
6-juin PM	info2303	ecrit
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)
17-juin PM	info2306	ecrit
18-juin PM	info2324	oral(2)
24-juin PM	info2322	oral(2)
27-juin PM	info2301	oral(3)

hamoir		
4-juin AM	info2102	oral(3)
5-juin PM	ielv2113	oral(1)
6-juin PM	info2101	ecrit
7-juin PM	info2107	ecrit
10-juin PM	info2112	oral(3)

hamoir		
13-juin AM	info2105	ecrit
14-juin AM	info2109	oral(3)
16-juin AM	info2110	oral(2)
18-juin AM	info2111	ecrit
19-juin AM	info2111	oral(2)
21-juin AM	info2105	oral(4)
23-juin AM	info2106	ecrit
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit

hick		
3-juin AM	info2201	ecrit
4-juin AM	info2204	ecrit
6-juin PM	info2304	oral(1)
11-juin PM	info2302	oral(1)
17-juin PM	info2306	ecrit

hynderick_de_ghelcke		
31-mai AM	info2211	ecrit
2-juin AM	info2226	ecrit
3-juin AM	info2201	ecrit
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
12-juin PM	info2208	oral(3)
14-juin PM	info2231	oral(1)
18-juin AM	info2224	ecrit
20-juin PM	info2211	oral(2)
23-juin AM	info2233	oral(3)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

jadoul		
4-juin AM	info2102	oral(3)
5-juin PM	ielv2113	oral(1)
6-juin PM	info2101	ecrit
7-juin PM	info2107	ecrit
10-juin PM	info2112	oral(3)
13-juin AM	info2105	ecrit
14-juin AM	info2109	oral(3)
16-juin AM	info2110	oral(2)
21-juin AM	info2105	oral(4)
23-juin AM	info2106	ecrit

jadoul		
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit

...

vilz		
2-juin AM	info2325	oral(1)
4-juin PM	info2324	oral(1)
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)
16-juin PM	info2303	ecrit
17-juin PM	info2306	ecrit
19-juin AM	info2322	oral(1)
21-juin PM	info2301	oral(1)

voogd		
4-juin PM	info2324	oral(1)
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)
16-juin AM	info2326	oral(1)
17-juin PM	info2306	ecrit
19-juin AM	info2322	oral(1)
21-juin PM	info2301	oral(1)
26-juin PM	info2303	ecrit

wathelet		
31-mai AM	info2211	ecrit
2-juin AM	info2226	ecrit
3-juin AM	info2201	ecrit
4-juin AM	info2211	oral(1)
5-juin PM	info2232	ecrit
7-juin AM	info2205	oral(1)
10-juin AM	info2210	oral(1)
11-juin AM	info2224	ecrit
13-juin PM	info2229	ecrit
16-juin PM	info2208	oral(1)
19-juin AM	info2233	oral(1)
26-juin AM	ielv2212	oral(1)
27-juin AM	info2202	ecrit
27-juin PM	info2202	oral

willame		
2-juin AM	info2325	oral(1)
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)
17-juin PM	info2306	ecrit
19-juin AM	info2322	oral(1)
21-juin PM	info2301	oral(1)
26-juin PM	info2303	ecrit
27-juin PM	info2327	oral(1)

willemyns		
2-juin AM	info2109	oral(1)
3-juin PM	info2110	oral(1)
5-juin PM	ielv2113	oral(1)
6-juin PM	info2101	ecrit
7-juin PM	info2107	ecrit
13-juin AM	info2105	ecrit
16-juin PM	info2112	oral(1)
18-juin AM	info2111	ecrit
19-juin AM	info2105	oral(1)
23-juin AM	info2106	ecrit
24-juin AM	info2111	oral(1)
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit
27-juin PM	info2102	oral(1)

zuyderhoff		
2-juin AM	info2325	oral(1)
11-juin PM	info2302	oral(1)
12-juin PM	info2301	ecrit
13-juin PM	ielv2307	oral(1)
16-juin AM	info2303	ecrit
17-juin PM	info2306	ecrit
19-juin AM	info2322	oral(1)
21-juin PM	info2301	oral(1)
27-juin PM	info2327	oral(1)

L'horaire a d'abord été calculé avec des contraintes plus importantes, notamment en ce qui concerne certains étudiants inscrits sur deux années. Ces étudiants devaient (théoriquement) présenter plus de vingt épreuves. Nous en fournissons quelques résultats ci-après, pour preuve que le programme se débrouille bien.

Session de juin 2003 (données non corrigées)

Quelques horaires individuels sont présentés afin de pouvoir vérifier la cohérence générale. L'horaire le plus intéressant est celui de l'étudiant *maes* qui devait présenter 27 épreuves.

achbany		
31-mai PM	info2211	ecrit
2-juin AM	info2208	oral(5)
6-juin AM	info2226	ecrit
10-juin AM	info2211	oral(5)
10-juin PM	info2205	oral(1)
12-juin PM	info2230	oral(6)
14-juin PM	info2210	oral(1)
17-juin PM	info2229	ecrit
24-juin AM	info2202	ecrit
24-juin PM	info2202	oral
25-juin AM	info2233	oral(6)
26-juin AM	ielv2212	oral(1)
26-juin PM	info2201	ecrit
27-juin AM	info2232	ecrit

lefevre		
4-juin AM	info2101	ecrit
4-juin PM	info2107	ecrit
5-juin PM	info2106	ecrit
6-juin PM	info2114	oral(1)
7-juin PM	info2109	oral(2)
12-juin AM	info2110	oral(2)
16-juin AM	info2102	oral(2)
18-juin AM	info2105	ecrit
19-juin PM	info2105	oral(3)
20-juin PM	ielv2113	oral(1)
24-juin AM	info2303	ecrit
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit

maes		
31-mai PM	info2211	ecrit
2-juin PM	info2208	oral(2)
4-juin AM	info2101	ecrit
4-juin PM	info2107	ecrit
5-juin PM	info2106	ecrit
6-juin AM	info2226	ecrit
6-juin PM	info2224	ecrit
7-juin PM	info2109	oral(2)
10-juin PM	info2205	oral(1)
11-juin AM	info2111	ecrit
12-juin AM	info2110	oral(2)
14-juin AM	info2111	oral(3)
14-juin PM	info2210	oral(1)

maes		
16-juin AM	info2102	oral(2)
16-juin PM	info2221	oral(1)
17-juin PM	info2112	oral(3)
18-juin AM	info2105	ecrit
19-juin PM	info2105	oral(3)
20-juin AM	info2211	oral(2)
20-juin PM	ielv2113	oral(1)
23-juin PM	info2233	oral(2)
24-juin AM	info2202	ecrit
24-juin PM	info2202	oral
25-juin AM	sent2288	ecrit
26-juin AM	info2108	ecrit
26-juin PM	info2201	ecrit
27-juin AM	info2232	ecrit

magnant		
31-mai PM	info2211	ecrit
2-juin PM	info2208	oral(2)
10-juin PM	info2205	oral(1)
11-juin AM	info2230	oral(2)
14-juin PM	info2210	oral(1)
16-juin PM	info2221	oral(1)
20-juin AM	info2211	oral(2)
23-juin PM	info2233	oral(2)
24-juin AM	info2202	ecrit
24-juin PM	info2202	oral
26-juin AM	ielv2212	oral(1)
26-juin PM	info2201	ecrit
27-juin AM	info2232	ecrit
27-juin PM	info2224	ecrit

Malgré le très grand nombre d'épreuves que doit passer l'étudiant *maes*, un horaire est établi. Dans de telles circonstances, les répercussions sur les horaires de certains autres étudiants peuvent être importantes. Ceux-ci ne paraissent cependant pas mauvais.

Réaliser un tel horaire manuellement semble pratiquement impossible.

Session d'août 2003 (samedis compris)

Un premier essai s'est inspiré de ce qui se faisait manuellement, à savoir, l'utilisation des samedis comme jours possibles d'examens.

HORAIRE DE LA SESSION		
Jours	1ère M	2e M
16-août AM		
16-août PM		info2210:ecrit
17-août AM		
17-août PM		
18-août AM	info2108:ecrit	
18-août PM		info2210:oral(1)
19-août AM	info2102:oral(1)	info2211:oral(1)
19-août PM		info2209:ecrit
20-août AM		
20-août PM	info2112:oral(1)	info2222:ecrit
21-août AM		
21-août PM		
22-août AM	info2106:ecrit	
22-août PM		info2226:ecrit
23-août AM		info2202:ecrit
23-août PM		info2202:oral
24-août AM		
24-août PM		
25-août AM	info2101:ecrit	info2230:oral(1)
25-août PM	info2111:ecrit	info2229:oral(1)
26-août AM	info2107:ecrit	info2204:ecrit
26-août PM	info2105:ecrit	
27-août AM	info2111:oral(1)	info2201:ecrit
27-août PM	info2105:oral(1)	info2201:oral(1)
28-août AM	ielv2113:ecrit	ielv2212:ecrit
28-août PM	ielv2113:oral	ielv2212:oral
29-août AM		info2201:oral(3)
29-août PM		info2232:ecrit
30-août AM	info2109:oral(2)	
30-août PM	info2110:oral(1)	
31-août AM		
31-août PM		
01-sept AM		sent2288:ecrit
01-sept PM	info2109:oral(1)	info2201:oral(4)
02-sept AM		info2201:oral(2) info2208:oral(1)
02-sept PM		info2233:oral(1)

Cet horaire a été établi en fonction des contraintes réelles. Quelques éléments peu déterminants nous manquaient toutefois. Leur connaissance aurait encore simplifié la conception de l'horaire. Par exemple, le professeur dont l'identifiant est *ielv* fait en réalité passer les épreuves écrites et orales des deux années sur un seul demi-jour au lieu d'un jour complet.

Voici quelques horaires individuels.

andre		
20-août PM	info2112	oral(1)
26-août AM	info2107	ecrit
28-août AM	ielv2113	ecrit
28-août PM	ielv2113	oral

biekonda_lonto		
18-août AM	info2108	ecrit
19-août AM	info2102	oral(1)
20-août PM	info2112	oral(1)
22-août AM	info2106	ecrit
25-août AM	info2101	ecrit
25-août PM	info2111	ecrit
26-août AM	info2107	ecrit
27-août AM	info2111	oral(1)
28-août AM	ielv2113	ecrit
28-août PM	ielv2113	oral
30-août AM	info2109	oral(2)
30-août PM	info2110	oral(1)
01-sept AM	sent2288	ecrit

bol		
16-août PM	info2210	ecrit
18-août PM	info2210	oral(1)
19-août AM	info2211	oral(1)
19-août PM	info2209	ecrit
23-août AM	info2202	ecrit
23-août PM	info2202	oral
25-août PM	info2229	oral(1)
26-août AM	info2204	ecrit
27-août AM	info2201	ecrit
28-août AM	ielv2212	ecrit
28-août PM	ielv2212	oral
29-août PM	info2232	ecrit
01-sept PM	info2201	oral(4)
02-sept AM	info2208	oral(1)
02-sept PM	info2233	oral(1)

canlas		
16-août PM	info2210	ecrit
18-août PM	info2210	oral(1)
19-août AM	info2211	oral(1)
19-août PM	info2209	ecrit
26-août AM	info2204	ecrit
27-août AM	info2201	ecrit

canlas		
29-août PM	info2232	ecrit
01-sept PM	info2201	oral(4)
02-sept AM	info2208	oral(1)
02-sept PM	info2233	oral(1)

...

tu		
20-août PM	info2112	oral(1)
01-sept AM	sent2288	ecrit

turine		
18-août AM	info2108	ecrit
19-août AM	info2102	oral(1)
20-août PM	info2112	oral(1)
22-août AM	info2106	ecrit
25-août AM	info2101	ecrit
26-août AM	info2107	ecrit
26-août PM	info2105	ecrit
27-août PM	info2105	oral(1)
28-août AM	ielv2113	ecrit
28-août PM	ielv2113	oral
30-août PM	info2110	oral(1)
01-sept AM	sent2288	ecrit
01-sept PM	info2109	oral(1)

van_tongelen		
18-août AM	info2108	ecrit
19-août AM	info2102	oral(1)
20-août PM	info2112	oral(1)
22-août AM	info2106	ecrit
25-août AM	info2101	ecrit
25-août PM	info2111	ecrit
26-août AM	info2107	ecrit
27-août AM	info2111	oral(1)
28-août AM	ielv2113	ecrit
28-août PM	ielv2113	oral
30-août PM	info2110	oral(1)
01-sept AM	sent2288	ecrit
01-sept PM	info2109	oral(1)

vanderwhale		
18-août AM	info2108	ecrit
22-août AM	info2106	ecrit
25-août AM	info2101	ecrit

vanderwhale		
26-août AM	info2107	ecrit
01-sept PM	info2109	oral(1)

Session d'août 2003 (samedis non compris sauf le premier)

HORAIRE DE LA SESSION		
Jours	1ère M	2e M
16-août AM	info2112:oral(1)	
16-août PM		info2210:ecrit
17-août AM		
17-août PM		
18-août AM	info2108:ecrit	
18-août PM		info2210:oral(1)
19-août AM		info2209:ecrit
19-août PM	info2102:oral(1)	info2222:ecrit
20-août AM	info2110:oral(1)	info2201:ecrit
20-août PM	info2109:oral(2)	info2201:oral(3)
21-août AM		info2211:oral(1)
21-août PM	info2105:ecrit	info2201:oral(1)
22-août AM	info2106:ecrit	info2201:oral(4)
22-août PM	info2109:oral(1)	info2226:ecrit
23-août AM		
23-août PM		
24-août AM		
24-août PM		
25-août AM		info2229:oral(1)
25-août PM		info2230:oral(1)
26-août AM	ielv2113:ecrit	ielv2212:ecrit
26-août PM	ielv2113:oral	ielv2212:oral
27-août AM	info2107:ecrit	info2204:ecrit
27-août PM		info2201:oral(2)
28-août AM		info2202:ecrit
28-août PM	info2111:ecrit	info2202:oral
29-août AM	info2105:oral(1)	
29-août PM	info2101:ecrit	
30-août AM		
30-août PM		
31-août AM		
31-août PM		
01-sept AM		info2232:ecrit sent2288:ecrit
01-sept PM		info2208:oral(1)
02-sept AM	info2111:oral(1)	info2233:oral(1)
02-sept PM		

L'horaire effectif, réalisé manuellement, inclut trois samedis et la journée du 3 septembre. Cette dernière figure dans l'horaire manuel parce qu'une contrainte est venue s'ajouter alors que l'horaire était déjà établi. Ceci montre bien la nécessité de disposer d'un maximum d'informations avant tout calcul d'horaire, au risque de devoir se satisfaire d'un horaire rafistolé.

Quelques horaires individuels...

andre		
16-août AM	info2112	oral(1)
26-août AM	ielv2113	ecrit
26-août PM	ielv2113	oral
27-août AM	info2107	ecrit

biekonda_lonto		
16-août AM	info2112	oral(1)
18-août AM	info2108	ecrit
19-août PM	info2102	oral(1)
20-août AM	info2110	oral(1)
20-août PM	info2109	oral(2)
22-août AM	info2106	ecrit
26-août AM	ielv2113	ecrit
26-août PM	ielv2113	oral
27-août AM	info2107	ecrit
28-août PM	info2111	ecrit
29-août PM	info2101	ecrit
01-sept AM	sent2288	ecrit
02-sept AM	info2111	oral(1)

bol		
16-août PM	info2210	ecrit
18-août PM	info2210	oral(1)
19-août AM	info2209	ecrit
20-août AM	info2201	ecrit
21-août AM	info2211	oral(1)
22-août AM	info2201	oral(4)
25-août AM	info2229	oral(1)
26-août AM	ielv2212	ecrit
26-août PM	ielv2212	oral
27-août AM	info2204	ecrit
28-août AM	info2202	ecrit
28-août PM	info2202	oral
01-sept AM	info2232	ecrit
01-sept PM	info2208	oral(1)
02-sept AM	info2233	oral(1)

canlas		
16-août PM	info2210	ecrit
18-août PM	info2210	oral(1)
19-août AM	info2209	ecrit
20-août AM	info2201	ecrit
21-août AM	info2211	oral(1)
22-août AM	info2201	oral(4)
27-août AM	info2204	ecrit
01-sept AM	info2232	ecrit
01-sept PM	info2208	oral(1)
02-sept AM	info2233	oral(1)

...

tu		
16-août AM	info2112	oral(1)
01-sept AM	sent2288	ecrit

---

turine		
16-août AM	info2112	oral(1)
18-août AM	info2108	ecrit
19-août PM	info2102	oral(1)
20-août AM	info2110	oral(1)
21-août PM	info2105	ecrit
22-août AM	info2106	ecrit
22-août PM	info2109	oral(1)
26-août AM	ielv2113	ecrit
26-août PM	ielv2113	oral
27-août AM	info2107	ecrit
29-août AM	info2105	oral(1)
29-août PM	info2101	ecrit
01-sept AM	sent2288	ecrit

---

van_tongelen		
16-août AM	info2112	oral(1)
18-août AM	info2108	ecrit
19-août PM	info2102	oral(1)
20-août AM	info2110	oral(1)
22-août AM	info2106	ecrit
22-août PM	info2109	oral(1)
26-août AM	ielv2113	ecrit
26-août PM	ielv2113	oral
27-août AM	info2107	ecrit
28-août PM	info2111	ecrit
29-août PM	info2101	ecrit
01-sept AM	sent2288	ecrit
02-sept AM	info2111	oral(1)

vanderwhale		
18-août AM	info2108	ecrit
22-août AM	info2106	ecrit
22-août PM	info2109	oral(1)
27-août AM	info2107	ecrit
29-août PM	info2101	ecrit

Ces résultats incluent un certain nombre de contraintes très fortes et notamment, des contraintes fixant le jour de certains examens. C'est le cas lorsqu'ils sont communs avec des examens de la *LIHD* et que le professeur souhaite faire passer les écrits correspondants au même moment. Cette situation se présente parfois en seconde session.

Les résultats qui suivent concernent les horaires individuels obtenus lors d'un premier essai qui n'incluait pas ces contraintes fortes. Ainsi peut-on juger de leur influence sur le résultat final.

Session d'août 2003 (sans contraintes fortes)

andre		
25-août AM	ielv2113	oral(1)
29-août AM	info2107	ecrit
02-septembre AM	info2112	oral(1)

biekonda_lonto		
16-août AM	info2109	oral(2)
19-août AM	info2111	ecrit
21-août PM	info2102	oral(1)
22-août PM	info2111	oral(1)
23-août PM	info2101	ecrit
25-août AM	ielv2113	oral(1)
28-août AM	sent2288	ecrit
29-août AM	info2107	ecrit
30-août AM	info2110	oral(1)
01-septembre AM	info2106	ecrit
02-septembre AM	info2112	oral(1)
03-septembre AM	info2108	ecrit

bol		
16-août PM	info2209	ecrit
18-août AM	info2210	ecrit
19-août AM	info2211	oral(1)
20-août AM	info2210	oral(1)
21-août AM	info2232	ecrit
22-août PM	info2201	ecrit
25-août AM	info2202	ecrit
25-août PM	info2202	oral
27-août PM	info2201	oral(4)
28-août PM	ielv2212	oral(1)

bol		
29-août PM	info2204	ecrit
01-septembre AM	info2233	oral(1)
02-septembre AM	info2208	oral(1)
03-septembre AM	info2229	oral(1)

canlas		
16-août PM	info2209	ecrit
18-août AM	info2210	ecrit
19-août AM	info2211	oral(1)
20-août AM	info2210	oral(1)
21-août AM	info2232	ecrit
22-août PM	info2201	ecrit
27-août PM	info2201	oral(4)
29-août PM	info2204	ecrit
01-septembre AM	info2233	oral(1)
02-septembre AM	info2208	oral(1)

...

tu		
28-août AM	sent2288	ecrit
02-septembre AM	info2112	oral(1)

turine		
20-août PM	info2109	oral(1)
21-août PM	info2102	oral(1)
23-août PM	info2101	ecrit
25-août AM	ielv2113	oral(1)
26-août AM	info2105	ecrit
27-août AM	info2105	oral(1)
28-août AM	sent2288	ecrit
29-août AM	info2107	ecrit
30-août AM	info2110	oral(1)
01-septembre AM	info2106	ecrit
02-septembre AM	info2112	oral(1)
03-septembre AM	info2108	ecrit

van_tongelen		
19-août AM	info2111	ecrit
20-août PM	info2109	oral(1)
21-août PM	info2102	oral(1)
22-août PM	info2111	oral(1)
23-août PM	info2101	ecrit
25-août AM	ielv2113	oral(1)
28-août AM	sent2288	ecrit

van_tongelen		
29-août AM	info2107	ecrit
30-août AM	info2110	oral(1)
01-septembre AM	info2106	ecrit
02-septembre AM	info2112	oral(1)
03-septembre AM	info2108	ecrit

vanderwhale		
20-août PM	info2109	oral(1)
23-août PM	info2101	ecrit
29-août AM	info2107	ecrit
01-septembre AM	info2106	ecrit
03-septembre AM	info2108	ecrit